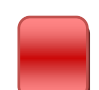


龙芯 2F 处理器用户手册



Revision 1.0

2008 年 8 月

北京龙芯中科技术服务中心有限公司

目 录

图目录

表目录

1 龙芯 2F 处理器微体系结构

1.1 龙芯系列微处理器介绍

龙芯处理器主要包括三个系列。龙芯 1 号处理器及其 IP 系列主要面向嵌入式应用，龙芯 2 号超标量处理器及其 IP 系列主要面向桌面应用，龙芯 3 号多核处理器系列主要面向服务器和高性能机应用。根据应用的需要，其中部分龙芯 2 号也可以面向部分高端嵌入式应用，部分低端龙芯 3 号也可以面向部分桌面应用。以后上述三个系列将并行地发展。

龙芯系列处理器通过充分开发指令级并行性、数据级并行性、线程级并行性来提高性能。其中龙芯 1 号系列微处理器实现了带有静态分支预测和阻塞 Cache 的单发射乱序执行流水线；龙芯 2 号系列微处理器实现了带有动态分支预测和非阻塞 Cache 的超标量四发射乱序执行流水线，龙芯 2 号系列微处理器还使用浮点数据通路复用技术实现了定点的单指令流多数据流指令；下一代的龙芯 3 号系列微处理器将实现片内多核技术。

1.2 龙芯 2F 处理器微体系结构概述

龙芯 2F 处理器是一款实现 64 位 MIPS III 指令集的高性能通用处理器芯片，片内集成 PCI/PCI-X 等 IO 控制器。龙芯 2F 的指令流水线每个时钟周期取四条指令进行译码，并且动态地发射到五个全流水的功能部件中。虽然指令在保证依赖关系的前提下进行乱序执行，但是指令的提交还是按照程序原来的顺序，以保证精确中断和访存顺序执行。

四发射的超标量结构使得指令流水线中指令和数据相关问题十分突出，龙芯 2F 采用乱序执行技术和激进的存储系统设计来提高流水线的效率。

乱序执行技术包括寄存器重命名技术、动态调度技术和转移预测技术。寄存器重命名解决 WAR（读后写）和 WAW（写后写）相关，并用于例外和错误转移预测引起的精确现场恢复，龙芯 2F 分别通过 64 项的物理寄存器堆进行定点和浮点寄存器的重命名。动态调度根据指令操作数准备好的次序而不是指令在程序中出现的次序来执行指令，减少了 RAW（写后读）相关引起的阻塞，龙芯 2F 有一个 16 项的定点保留站和一个 16 项的浮点保留站用于乱序发射，并通过一个 64 项的 Reorder 队列（简称 ROQ）实现乱序执行的指令按照程序的次序提交。转移预测通过预测转移指令是否成功跳转来减少由于控制相关引起的阻塞，龙芯 2F 使用 16 项的转移目标地址缓冲器（Branch Target Buffer，简称 BTB），2K 项的转移历史表（Branch History Table，简称 BHT），9 位的全局历史寄存器（Global Histiry Registor，简称 GHR），和 4 项的返回地址栈（Return Address Stack，简称 RAS）进行转移预测。

龙芯 2F 先进的存储系统设计可以有效地提高流水线的效率。龙芯 2F 的一级 Cache 由 64KB 的指令 Cache 和 64KB 的数据 Cache 组成，片上二级 Cache 大小为 512KB，均

采用四路组相联的结构。龙芯 2F 处理器内部集成了遵守 JESD79-2B 标准的 DDR2 控制器，加快了处理器访问内存的速度。龙芯 2F 的 TLB 有 64 项，采用全相联结构，每项可以映射一个奇页和一个偶页。龙芯 2F 通过 24 项的访存队列以及 8 项的访存失效队列（Miss Queue）来动态地解决地址依赖，实现访存操作的乱序执行、非阻塞 Cache、取数指令猜测执行（Load Speculation）、写合并（Store Fill Buffer）等访存优化技术。

龙芯 2F 有两个定点功能部件和两个浮点功能部件。浮点部件通过浮点指令的 FMT 域的扩展可以执行 32 位和 64 位的定点指令，以及 8 位和 16 位的用于媒体加速的 SIMD 指令。

龙芯 2F 的基本流水线包括取指、预译码、译码、寄存器重命名、调度、发射、读寄存器、执行、提交等 9 级，每一级流水包括如下操作。

- **取指**流水级用程序计数器 PC 的值去访问指令 Cache 和指令 TLB，如果指令 Cache 和指令 TLB 都命中，则把四条新的指令取到指令寄存器 IR。
- **预译码**流水级主要对转移指令进行译码并预测跳转的方向。
- **译码**流水级把 IR 中的四条指令转换成龙芯 2F 的内部指令格式送往寄存器重命名模块。
- **寄存器重命名**流水级为逻辑目标寄存器分配一个新的物理寄存器，并将逻辑源寄存器映射到最近分配给该逻辑寄存器的物理寄存器。
- **调度**流水级将重命名的指令分配到定点或浮点保留站中等待执行，同时送到 ROQ 中用于执行后的顺序提交；此外，转移指令和访存指令还分别被送往转移队列和访存队列。
- **发射**流水级从定点或浮点保留站中为每个功能部件选出一条所有操作数都准备好的指令；在重命名时操作数没准备好的指令，通过侦听结果总线和 Forward 总线等待它的操作数准备好。
- **读寄存器**流水级为发射的指令从物理寄存器堆中读取相应的源操作数送到相应的功能部件。
- **执行**流水级根据指令的类型执行指令并把计算结果写回寄存器堆；结果总线还送往保留站和寄存器重命名表，通知相应的寄存器值已经可以使用了。
- **提交**流水级按照 Reorder 队列记录的程序的顺序提交已经执行完的指令，龙芯 2F 最多每拍可以提交四条指令，提交的指令送往寄存器重命名表用于确认它的目的寄存器的重命名关系并释放原来分配给同一逻辑寄存器的物理寄存器，并送往访存队列允许那些提交的存数指令写入 Cache 或内存。

上述是基本指令的流水级，对于一些较复杂的指令，如定点乘除法指令、浮点指令以及访存指令，在执行阶段需要多拍。龙芯 2F 处理器的基本结构如图 1-1 所示。

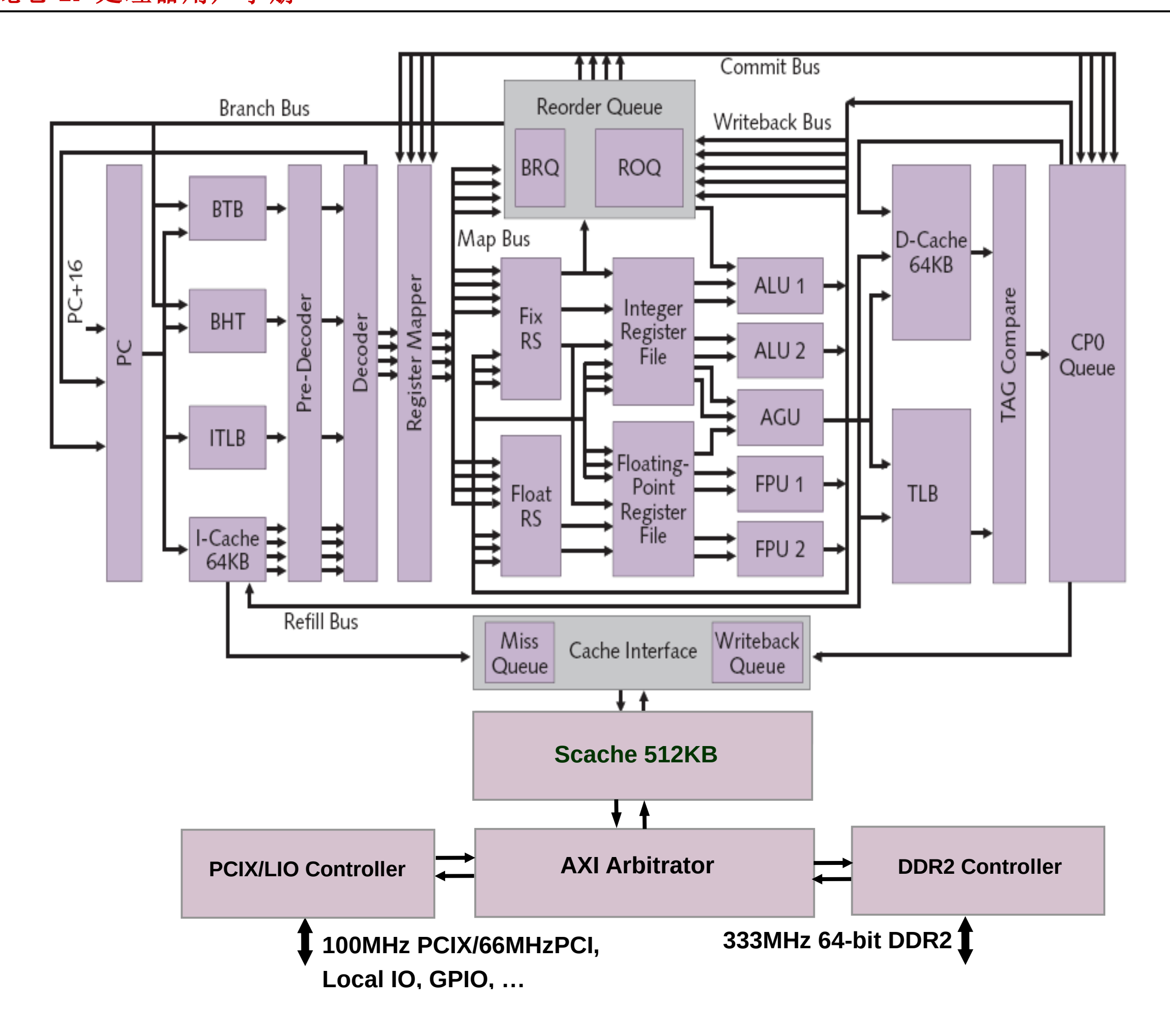


图 1-1 龙芯 2F 体系结构框图

1.3 取指和分支预测

龙芯 2F 的流水线从取指流水级开始，每次取四条指令，但每次取指不能跨越 32 字节的指令 Cache 行。龙芯 2F 取指时同时访问指令 Cache 和指令 TLB（简称 ITLB）。为了降低延迟，Tag 比较在取指阶段进行，但根据 Tag 比较结果进行指令选择在预译码阶段进行。取指过程中发生指令 Cache 不命中时向二级 Cache 发出访问请求。

16 项的 ITLB 是主 TLB 的子集。当 ITLB 不命中时，龙芯 2F 产生一个内部指令去查找主 TLB 并且填充 ITLB，如果在主 TLB 中也不命中将产生一个普通的 TLB 例外。

取指后的流水级是预译码流水级。这一级的主要工作是预测转移指令的跳转方向以及目标地址。不同的转移指令使用不同的方式进行预测：Likely 类转移指令和直接跳转指令总是被预测为跳转，编译器可以通过编译出 Likely 指令进行静态预测；条件转移指令通过 BHT 预测跳转方向；间接跳转指令则用转移目标表（BTB）或返回地址栈（RAS）预测目标地址。

BHT 包括一个 9 位的全局历史寄存器（GHR）和一个 2K 项的模式历史表（PHT）。

PHT 的每项是一个两位的饱和计数器，预测正确时计数器加 1，预测错误时计数器减 1。当计数器的值大于等于 2 时预测跳转成功。

16 项的 BTB 用于预测寄存器跳转指令的目标地址。每项 BTB 保存转移指令的地址和目标地址，以及一个两位的饱和计数器。当发生替换时，计数器的值小于 2 的项会优先被替换。

MIPS 指令集中没有 Call 和 Return 指令，通常使用转移链接（Jump and Link）指令和 jr31 指令进行函数调用和返回。龙芯 2F 实现 4 项的返回地址栈 RAS。当译码出转移链接指令时将它的 PC＋8 压入 RAS，当译码出 jr31 指令时弹出 RAS 的顶作为 jr31 的目标地址。

龙芯 2F 的第三级流水级是译码流水级。在这一级，四条指令被译成龙芯 2F 的内部指令格式送往寄存器重命名模块。由于定点乘法指令和定点除法指令要生成两个 64 位结果，所以被译为两条内部指令。为了简化转移指令的管理，龙芯 2F 每拍最多只进行一条转移指令的译码。

1.4 寄存器重命名

龙芯 2F 使用物理寄存器堆的方法进行寄存器重命名，其中定点和浮点物理寄存器堆各为 64 项。龙芯 2F 通过两个 64 项的物理寄存器映射表（Physical Register Mapping Table, 简称 PRMT）来保存物理寄存器和结构寄存器间的映射关系。

龙芯 2F 的每个物理寄存器都处于以下四个状态的其中一个：MAP_EMPTY 表示该物理寄存器没有使用，MAP_MAPPED 表示该物理寄存器已经被映射但相应的值没有写回，MAP_WTBK 表示该物理寄存器的值已经写回，MAP_COMMIT 表示该物理寄存器值已经确定为处理器状态。

在寄存器重命名流水级，每条指令通过查找 PRMT 表得到该指令的两个源寄存器 SRC1、SRC2，和一个目标寄存器 DEST 当前所对应的物理寄存器号 PSRC1，PSRC2 和 ODEST。同时为目标寄存器 DEST 分配一个状态为 MAP_EMPTY 的一个物理寄存器 PDEST，新分配的物理寄存器的状态改为 MAP_MAPPED。同时修改 PRMT 表示 PDEST 是结构寄存器 DEST 的最新的映射。

在查找 PRMT 表建立逻辑寄存器和物理寄存器之间映射关系的同时，还需要检查同一拍重命名的四条指令间的相关性。如果某条指令 A 的源寄存器 SRC1 和同一拍前面指令 B 的目的寄存器 DEST 相同，则 A 的 SRC1 对应的物理寄存器改为 B 新分配的 PDEST，而非 A 从 PRMT 中查出的 PSRC1。相同的原则也适用 PSRC2 和 ODEST。

经过寄存器重命名，物理寄存器号 PSRC1，PSRC2 和 PDEST 替换了原来指令中的结构寄存器号 SRC1，SRC2 和 DEST。其中物理寄存器号 PSRC1 和 PSRC2 送到保留站用于判断指令间的数据相关；ODEST 域保存在 ROQ 中，在指令提交时用于释放物理寄存器。

指令执行时，该指令的 PDEST 对应的 PRMT 的项设为 MAP_WTBK，表示该寄存器的值已经准备好了，后面的指令可以使用该寄存器的值。

指令提交时，该指令的 PDEST 对应的 PRMT 项设为 MAP_COMMIT 状态，ODEST 对应的 PRMT 项设为 MAP_EMPTY 状态，表示为该指令新分配的目的寄存器 PDEST 成为处理器状态，并释放该指令的目标寄存器原来所对应的物理寄存器。

从上面的寄存器重命名的过程可以看出一个结构寄存器可能同时对应多个物理寄存器，即一个逻辑寄存器在流水线中由于被多条指令修改可能有一系列的值。与一个结构寄存器对应的多个物理寄存器除了有一个表示该结构寄存器的处理器状态外，其它的分别对应于在流水线中的写该逻辑寄存器的多条指令。每个物理寄存器在每次分配之后只会被写一次。

1.5 指令发射和读寄存器

寄存器重命名后的指令送到保留站调度执行。龙芯 2F 具有两个独立的分组保留站：定点和访存指令送到定点保留站；浮点指令送到浮点保留站。每一个保留站 16 项。

在寄存器重命名阶段，每条指令查找 PRMT 表确定操作数是否在寄存器堆中。如果查找 PRMT 表时相应的操作数没有准备好，该指令在送入保留站的途中以及在保留站中要通过比较自己的源寄存器号和结果总线或 Forward 总线的目标寄存器号以确定源操作数何时准备好。结果总线和 Forward 总线来自五个功能部件，结果总线送出指令的执行结果以及目标寄存器号，而 Forward 总线预测下一拍会被送出的结果以及相应指令的目标寄存器号。

两个保留站每拍最多可以发射五个源操作数准备好的指令到五个功能部件。如果在保留站中同一个功能部件有多个操作数准备好的指令，则选择最“老”的指令进行发射。在保留站中用一个 AGE 域来记录每一条指令在保留站中的“年龄”。

从保留站发射的指令到寄存器堆中读操作数后再送到功能部件执行。龙芯 2F 有一个定点寄存器堆和一个浮点寄存器堆，大小都是 64*64。定点寄存器堆有 3 个写端口和 7 个读端口，其中 ALU1 使用 1 个写端口和 3 个读端口，ALU2 和访存部件各使用 1 个写端口和 2 个读端口。浮点寄存器堆有 3 个写端口和 7 个读端口，其中两个浮点部件各使用 1 个写端口和 3 个读端口，访存部件使用 1 个写端口和 1 个读端口用于浮点取数和存数指令。定点和浮点寄存器间的数据传输指令，如 MTC1、DMTC1、MFC1、DMFC1、CTC1 和 CFC1 使用访存数据通路传输数据，因此由访存部件执行。

特殊指令如 Branch and Link 指令的程序计数器或条件转移指令的 Taken 位从转移队列中读出并且与寄存器堆中的操作数一起送到相应的功能部件。

1.6 指令执行和功能部件

指令从寄存器堆中读取操作数后根据指令的类型送到相应的功能部件或访存部件执

行，龙芯 2F 包括两个定点部件 ALU1 和 ALU2，两个浮点部件 FALU1 和 FALU2。

定点 ALU1 执行定点加减、逻辑运算、移位、比较、Trap、以及转移指令。所有 ALU1 执行的指令 1 拍完成执行并写回。

定点 ALU2 执行定点加减、逻辑运算、移位、比较、以及乘除指令。定点乘法为全流水操作，延迟为 4 拍；定点除法采用 SRT 算法，非全流水操作，延迟根据操作数的不同从 4 拍到 37 拍不等；所有其他 ALU2 执行的指令 1 拍完成执行并写回。

浮点 FALU1 执行浮点加减、浮点乘法、浮点乘加（减）、取绝对值、取反、精度转换、定浮点格式转换、比较、转移等指令。FALU1 的所有运算为全流水操作。其中浮点取绝对值、取反、精度转换、比较、转移延迟为 2 拍，定浮点间格式转换延迟为 4 拍，浮点加减、浮点乘法、浮点乘加（减）延迟为 6 拍。

浮点 FALU2 执行浮点加减、浮点乘法、浮点乘加（减）、浮点除法、浮点开平方操作。其中浮点加减、浮点乘法、浮点乘加（减）为全流水操作，延迟为 6 拍；浮点除法和浮点开平方使用 SRT 算法，为非全流水操作，根据操作数的不同，单/双精度浮点除法延时从 4 到 10/17 拍不等，单/双精度开方延时从 4 到 16/31 拍不等。

除了执行 MIPS III 浮点指令外，浮点功能部件还可以执行并行单精度浮点指令，即在 64 位数据通路上同时计算两个单精度操作（加、减、乘和乘加）。另外，浮点功能部件还通过扩展浮点指令的格式域（FMT）执行 8/16/32/64 位 SIMD 多媒体定点指令。

1.7 指令提交和 Reorder 队列

在龙芯 2F 中，指令顺序译码和重命名，乱序发射和执行，但有序结束。Reorder 队列（Reorder Queue，简称 ROQ）负责指令的有序结束，它按照程序次序保存流水线中所有已经完成寄存器重命名但未提交的指令。指令执行完并写回后，ROQ 按照程序次序提交这些指令。ROQ 最多可以同时容纳 64 条指令。

每条完成寄存器重命名的指令在送入保留站的同时也送入 ROQ。新进入的指令置为 ROQ_MAPPED 状态。指令写回后，ROQ 中的普通指令置为 ROQ_WTBK，转移指令置为 ROQ_BRWTBK 状态。状态为 ROQ_BRWTBK 的转移指令通过转移总线送到处理器的其他部分根据转移指令的执行结果修正转移猜测表并在转移猜测错误的情况下取消转移指令及其后续指令，并把状态置为 ROQ_WTBK。ROQ_WTBK 状态的指令在成为 ROQ 的队列头时可以提交。

ROQ 一拍最多可以提交队列头上的四条 ROQ_WTBK 状态指令。提交指令的 PDEST 和 ODEST 域送到寄存器重命名模块确认 PDEST 项的重命名为处理器状态并释放 ODEST 项的映射，它还通知访存队列相应的 Store 指令可以开始修改存储器。

为了实现精确例外，在指令执行过程中发生例外时把例外原因记录在 ROQ 相应的项中。当例外指令成为 ROQ 的队列头时进行例外处理，把例外原因、例外指令的 PC 值等例外信息记录到有关的 CP0 寄存器中，并根据例外类型把例外处理程序的入口地址送到

程序计数器 PC 中。

1.8 转移取消和转移队列

转移指令在重命名后进入 ROQ 和保留站的同时进入转移队列。转移队列同时可以容纳多达 8 条转移指令。

当转移指令发射执行时，转移队列提供该指令执行所需的信息，这些信息包括转移指令的 PC 值，和条件转移指令的预测 Taken 位等。

转移指令执行后，结果写回到转移队列。这些结果包括 JR 和 JALR 指令的目标地址、条件转移指令的转移方向和转移指令是否预测错误的标志位。转移指令的执行结果在提交前通过转移总线反馈到取指部分用来修正 BHT、BTB、RAS 和 GHR 以进行接下来的转移预测。

预测错误的指令和它后面的指令都需要取消。转移取消的一个核心问题是如何判断在流水线中乱序执行的指令哪些在取消的转移操作之前，哪些在取消的转移操作之后。龙芯 2F 用转移指令把连续的指令流分为独立的基本块，并用转移指令在转移队列中的位置标识号 BRQID 对基本块进行编号。对于转移指令，这个标识表示它在转移队列中的位置；对于普通操作，这个标识表示它前面的转移指令在转移队列中的位置。通过这种方式，每一条指令都可以通过比较自己的 BRQID 和预测错误转移指令的 BRQID 确定它相对转移预测错误指令的位置。

1.9 存储访问与存储管理

龙芯 2F 存储子系统对提高处理器的流水线效率起着重要作用。龙芯 2F 一级指令和数据 Cache 大小均为 64KB，二级 Cache 大小为 512KB，均采用四路组相联结构；龙芯 2F 片内集成了 DDR 内存控制器接口；龙芯 2F 的 TLB 共有 64 项，为全相联结构，每项映射一个奇数页和一个偶数页；龙芯 2F 通过一个 24 项的访存队列和一个 8 项的失效队列来动态解决存储相关，实现访存指令乱序执行、非阻塞 Cache、Load 猜测执行和写合并等。

龙芯 2F 访存流水线分为 4 级。发射流水级把访存请求发射到地址运算部件后，第一拍通过地址运算部件计算虚地址并把访存请求送到 TLB 和 Cache；第二拍在 TLB 把虚地址转换为物理地址的同时访问 Cache；第三拍根据 TLB 和 Cache 的访问结果确定 Cache 是否命中并送到访存队列；第四拍把访问结果写回。

龙芯 2F 的存储系统使用 40 位虚地址和 40 位物理地址，并通过一个全相联的 TLB 进行虚实地址转换。该 TLB 包含一个 CAM 部分进行虚地址的全相联查找以及一个 RAM 部分存储物理页号和页的保护位。龙芯 2F 的 TLB 有 64 项，采用全相联结构，每项可以映射一个奇页和一个偶页。龙芯 2F 的 TLB 的一个重要特点是它的页执行保护功能，它是通过在 TLB 的每一项增加一个执行保护位来实现的。该位可以由软件进行设置，表示

相应的页是否可以被执行。硬件在取指过程中访问 TLB 时，除了做常规的权限检查外还进行可执行检查，如果取指时相应的页被置为不可执行，就会发生地址错例外。在操作系统中，只要利用上述方法对堆栈所在地址空间进行取指保护，就可以有效防范大多数利用缓冲区溢出技术进行的非法攻击。

龙芯 2F 的一级数据 Cache 有 64KB，为四路组相联结构，块大小为 32 字节，采用随机替换算法。该 Cache 采用虚地址 Index 以及物理地址 Tag 以进行并行的 Cache 和 TLB 查找。由于 Cache 的每一路包含 16K 字节（是最小虚页的 4 倍），虚地址 Index 的两位（13:12）可能与物理地址 Tag 的相应位不相等，操作系统需要通过页着色（Page Coloring）或增加页的大小（每页 16KB 以上）来解决虚地址 Index 引起的不一致问题。龙芯 2F Cache 的数据和标志部分都采用单端口 RAM。为了降低 Cache 访问冲突，龙芯 2F 把大小为 512*256 位的的每一路 Cache 分为四个 512*64 位的体，并允许对不同体的读和写同时进行以降低 Cache 访问冲突。在 Cache 失效时的回填操作 Refill、地址运算后访问 Cache、以及存数操作提交后的写回三种操作访问 Cache 端口冲突时，Refill 具有最高的优先级，存数操作的写回具有最低的优先级。

访存队列是龙芯 2F 存储子系统的核心部件。它记录最多 24 个未执行完的 Load 或 Store 操作。虽然 Load 和 Store 操作乱序进入队列，但在访存队列中按它们程序中出现的次序排列。访存队列允许 Cache 失效的访存操作后面的多个 Cache 失效或命中的访存操作继续进行。龙芯 2F 在 Cache 失效或访存相关时不重新进行访存，访存队列通过物理地址的全相联比较动态解决访存操作间的相关。取数操作进入队列时，通过地址比较按字节接收它前面的最近一个对同一地址的存数操作的值；存数操作进入队列时，通过地址比较按字节把所存的值传递给它后面对同一地址的取数操作，直到下一个对同一地址的存数操作。

一级 Cache 失效的取指或访存操作或被送入失效队列（Miss Queue）。失效队列处于指令 Cache、数据 Cache、二级 Cache、DDR 内存控制器接口、以及 SysAD 系统总线接口之间。该队列接收一级 Cache 失效的取指或访存操作并访问二级 Cache，并把相应的访问结果通过回填总线送回一级 Cache；在二级 Cache 访问失效时访问下一级存储器或系统总线接口，并把相应的访问结果送回二级以及一级 Cache。龙芯 2F 的失效队列还支持失效写合并（Store Fill Buffer）优化，即可以把多个对同一 Cache 块的写请求合并在一起组成完整的 Cache 块，避免了没必要的存储器访问。

龙芯 2F 集成了片上二级 Cache，二级 Cache 的块大小为 32 字节，容量 512KB，采用四路组相联结构。龙芯 2F 的 512KB 二级 Cache 由 64 个 1024*64 位的 RAM 块组成，每次访问二级 Cache 时，只要打开相应 RAM 块的片选使能以降低功耗。

龙芯 2F 内部集成的内存控制器的设计遵守 DDR2 SDRAM 的行业标准（JESD79-2B），支持最大 4 个物理内存 Bank（由 4 个 DDR2 SDRAM 片选信号实现），一共含有 15 位的地址总线（13 位行列地址总线和 2 位逻辑 Bank 总线）。龙芯 2F 内部集成的内存控制器

实现了一种动态的 Page 管理策略，针对一次访存操作，内存控制器对 Open Page 策略/Close Page 策略的选择是由硬件电路来实现的，无需软件设计人员来干预。

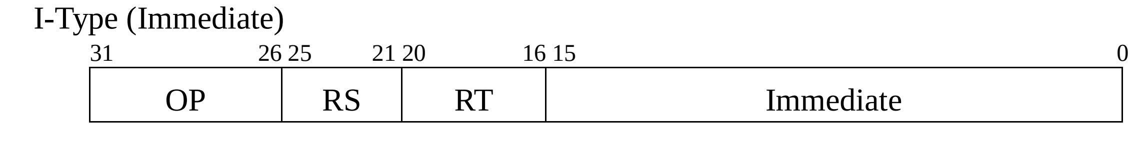
1.10 龙芯 2F 结构小结

龙芯 2F 是一款 64 位、四发射、乱序执行的 RISC 处理器，实现 MIPS III 指令集。该处理器采用先进的乱序执行技术（如寄存器重命名、转移预测和动态调度）和 Cache 技术（如非阻塞 Cache、load 猜测、动态内存相关和写合并技术），并集成片上二级 Cache、DDR2 内存控制器和 IO 控制器，来提高流水线效率和 IO 能力。

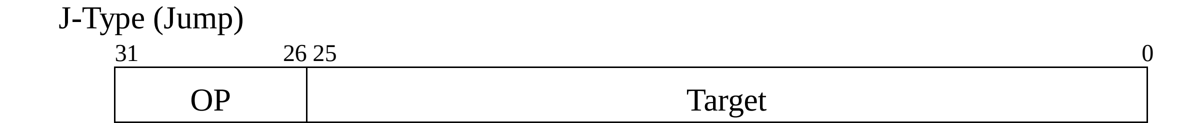
2 龙芯 2F 处理器指令集概述

每条 CPU 指令都是一条 32 位的指令字，这些指令都是字对齐的。指令集包含三种指令格式，如图 2-1 所示，立即数指令（I-型），跳转指令（J-型）和寄存器指令（R-型）。使用简单几种指令格式可以简化指令译码，并且使得编译器根据这三种指令格式可以合成更多的复杂操作（使用频率较低）和访存模式。

I-Type (Immediate)



J-Type (Jump)



R-Type (Register)

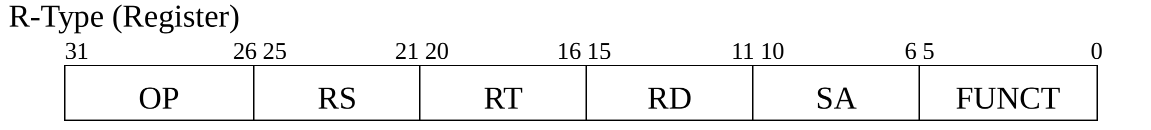


图 2-1 CPU 指令格式

OP	6 位操作码
RS	5 位源操作寄存器域
RT	5 位目标（源/目的）操作寄存器或跳转条件
Immediate	16 位立即数
Target	26 位跳转目标地址
RD	5 位目的操作寄存器域
SA	5 位移位位数
FUNCT	6 位功能域

指令集可以更进一步分为以下几组：

- **Load and Store** 访存指令在主存和通用寄存器之间移动数据。访存指令都是立即数指令（I-型），因为该指令模式所支持的唯一访存模式就是基址寄存器加上 16 位的对齐的偏移量。

- **Computational** 计算型指令完成寄存器值的算术、逻辑、移位、乘法和除法操作。计算型指令包含了寄存器指令格式（R-型，操作数和运算结果均保存在寄存器中）和立即数指令格式（I-型，其中一个操作数为一个 16 位的立即数）。龙芯 2F 微处理器还实现了自定义的乘法，除法和模操作指令，其方法是使用一个通用的目的寄存器来取代成对

出现的 HI 和 LO 寄存器。

- **Jump and Branch** 跳转和分支指令改变程序的控制流。绝对地址跳转被称为“Jump（跳转）”（J-型或者 R-型），PC（指令计数器）相关的跳转指令被称为“Branch（分支）”（I-型）。跳转指令的返回地址保存在第 31 号寄存器中。
- **Coprocessor** 协处理器指令完成协处理器内部的操作。协处理器的访存操作是 I-型指令。龙芯 2F 微处理器有两个协处理器：0 号协处理器（系统处理器）和 1 号协处理器（浮点协处理器）。

0 号协处理器（CP0）通过 CP0 的寄存器来管理内存和处理异常。这些指令列在表 2-9 中。

1 号协处理器（CP1）指令包括浮点指令，多媒体指令，和龙芯扩展的定点计算指令。这些指令都是在浮点寄存器上操作。第八章将会对这些指令进行总结，附录将会对每条指令进行详细的描述。

- **Special** 特殊指令完成系统调用和断点操作。这些指令通常是 R-型的。
- **Exception** 异常指令引起跳转，根据异常号比较结果跳转到通用异常处理向量。这些指令包括 R-型和 I-型指令格式。

表 2-1 到表 2-9 列出了除 1 号协处理器指令以外的所有指令。

表 2-1 CPU 指令集：访存指令

OpCode	Description	MIPS ISA
LB	取字节	I
LBU	取无符号字节	I
LH	取半字	I
LHU	取无符号半字	I
LW	取字	I
LWU	取无符号字	I
LWL	取字左部	I
LWR	取字右部	I
LD	取双字	III
LDL	取双字左部	III
LDR	取双字右部	III
LL	取标志处地址	I
LLD	取标志处双字地址	III
SB	存字节	I
SH	存半字	I
SW	存字	I
SWL	存字左部	I

OpCode	Description	MIPS ISA
SWR	存字右部	I
SD	存双字	III
SDL	存双字左部	III
SDR	存双字右部	III
SC	满足条件下存	I
SCD	满足条件下存双字	III
SYNC	同步	I

表 2-2 CPU 指令集：算术指令 (ALU 立即数)

OpCode	Description	MIPS ISA
ADDI	加立即数	I
DADDI	加双字立即数	III
ADDIU	加无符号立即数	I
DADDIU	加无符号双字立即数	III
SLTI	小于立即数设置	I
SLTIU	无符号小于立即数设置	I
ANDI	与立即数	I
ORI	或立即数	I
XORI	异或立即数	I
LUI	取立即数到高位	I

表 2-3 CPU 指令集：算术指令(3 操作数, R-型)

OpCode	Description	MIPS ISA
ADD	加	I
DADD	双字加	III
ADDU	无符号加	I
DADDU	无符号双字加	III
SUB	减	I
DSUB	双字减	III
SUBU	无符号减	I
DSUBU	无符号双字减	III
SLT	小于设置	I
SLTU	无符号小于设置	I
AND	与	I
OR	或	I

XOR	异或	I
NOR	或非	I

表 2-4 CPU 指令集：乘法和除法指令

OpCode	Description	MIPS ISA
MULT	乘	I
DMULT	双字乘	III
MULTU	无符号乘	I
DMULTU	无符号双字乘	III
DIV	除	I
DDIV	双字除	III
DIVU	无符号除	I
DDIVU	无符号双字除	III
MFHI	从 hi 寄存器取数到通用寄存器	I
MTHI	从通用寄存器存数到 hi 寄存器	I
MFLO	从 lo 寄存器取数到通用寄存器	I
MTLO	从通用寄存器存数到 lo 寄存器	I
MULTG	龙芯 2F 乘	GODSON2
DMULTG	龙芯 2F 双字乘	GODSON2
MULTUG	龙芯 2F 无符号乘	GODSON2
DMULTUG	龙芯 2F 无符号双字乘	GODSON2
DIVG	龙芯 2F 除	GODSON2
DDIVG	龙芯 2F 双字除	GODSON2
DIVUG	龙芯 2F 无符号除	GODSON2
DDIVUG	龙芯 2F 无符号双字除	GODSON2
MODG	龙芯 2F 求模	GODSON2
DMODG	龙芯 2F 双字求模	GODSON2
MODUG	龙芯 2F 无符号求模	GODSON2
DMODUG	龙芯 2F 无符号双字求模	GODSON2

表 2-5 CPU 指令集：跳转和分支指令

Opcode	Description	MIPS ISA
J	跳转	I
JAL	立即数调用子程序	I
JR	跳转到寄存器指向的指令	I
JALR	寄存器调用子程序	I

Opcode	Description	MIPS ISA
BEQ	相等则跳转	I
BNE	不等则跳转	I
BLEZ	小于等于 0 跳转	I
BGTZ	大于 0 跳转	I
BLTZ	小于 0 跳转	I
BGEZ	大于或等于 0 跳转	I
BLTZAL	小于 0 调用子程序	I
BGEZAL	大于或等于 0 调用子程序	I
BEQL	相等则 Likely 跳转	II
BNEL	不等则 Likely 跳转	II
BLEZL	小于或等于 0 则 Likely 跳转	II
BGTZL	大于 0 则 Likely 跳转	II
BLTZL	小于 0 则 Likely 跳转	II
BGEZL	大于或等于 0 则 Likely 跳转	II
BLTZALL	小于 0 则 Likely 调用子程序	II
BGEZALL	大于或等于 0 则 Likely 调用子程序	II

表 2-6 CPU 指令集：移位指令

OpCode	Description	MIPS ISA
SLL	逻辑左移	I
SRL	逻辑右移	I
SRA	算术右移	I
SLLV	可变的逻辑左移	I
SRLV	可变的逻辑右移	I
SRAV	可变的算术右移	I
DSLL	双字逻辑左移	III
DSRL	双字逻辑右移	III
DSRA	双字算术右移	III
DSLLV	可变的双字逻辑左移	III
DSRLV	可变的双字逻辑右移	III
DSRAV	可变的双字算术右移	III
DSLL32	双字逻辑左移+32	III
DSRL32	双字逻辑右移+32	III
DSRA32	双字算术右移+32	III

表 2-7 CPU 指令集：特殊指令

OpCode	Description	MIPS ISA
SYSCALL	系统调用	I
BREAK	断点	I

表 2-8 CPU 指令集：异常指令

OpCode	Description	MIPS ISA
TGE	大于或等于陷入	II
TGEU	无符号数大于或等于陷入	II
TLT	小于陷入	II
TLTU	无符号数小于陷入	II
TEQ	等于陷入	II
TNE	不等陷入	II
TGEI	大于或等于立即数陷入	II
TGEIU	大于或等于无符号立即数陷入	II
TLTI	小于立即数陷入	II
TLTIU	小于无符号立即数陷入	II
TEQI	等于立即数陷入	II
TNEI	不等于立即数陷入	II

表 2-9 CPU 指令集：CP0 指令

OpCode	Description	MIPS ISA
DMFC0	从 CP0 寄存器取双字	III
DMTC0	往 CP0 寄存器写双字	III
MFC0	从 CP0 寄存器取	I
MTC0	往 CP0 寄存器写	I
TLBR	读索引的 TLB 项	III
TLBWI	写索引的 TLB 项	III
TLBWR	写随机的 TLB 项	III
TLBP	在 TLB 中搜索匹配项	III
CACHE	Cache 操作	III
ERET	异常返回	III

3 内存管理

龙芯 2F 处理器提供了一个功能完备的内存管理单元（MMU），它利用片上的 TLB 实现虚拟地址到物理地址的转换。

本章节描述了处理器的虚拟地址和物理地址空间，虚拟地址到物理地址的转换，TLB 在实现这些转换时的操作，Cache，以及为 TLB 提供软件接口的系统控制协处理器（CP0）寄存器。

3.1 快速查找表 TLB

把虚拟地址映射成物理地址是由 TLB 来实现的。第一级的 TLB 是 JTLB，同时也作为数据 TLB，另外，龙芯 2F 处理器包含独立的指令 TLB 以缓解对 JTLB 的竞争。

3.1.1 JTLB

为了能够快速地进行虚拟地址到物理地址的映射，龙芯 2F 处理器采用了较大的，全相联映射机制的 TLB，JTLB 用于指令和数据的地址映射，用它们的名字进行索引。

JTLB 按奇/偶表项成对组织，把虚拟地址空间和地址空间标识符映射到 1T 的物理地址空间。在默认的情况下，JTLB 有 64 对奇/偶表项，允许 128 页进行映射。

有两个机制分别用来协助控制映射空间的大小和内存不同区域的替换策略。

第一，页的大小可以是 4KB 到 16MB，但必须是按 4 倍递增。CP0 寄存器 PageMask 用于记录映射的页的大小，并且这个记录在写一个新的表项的同时载入 TLB 中。因此操作系统可以支持不同大小的页表项以适用于不同的目的，然而在同一运行的时刻只能是固定大小的页。龙芯 2F 处理器在将来可以在同一运行时刻支持不同大小的页，允许操作系统产生特定目的的映射：例如，帧缓冲区就可以只用一个表项来进行内存映射。

第二，龙芯 2F 处理器在 TLB 缺失的时候采用随机替换的策略来选择要被替换的 TLB 表项。

也有不经过 TLB 的虚拟地址转换，比如 CKSEG0 和 CKSEG1 内核地址空间段（见图 3-5）就是不进行页面映射的，其中的物理地址是由虚拟地址减去一个基址得到的。操作系统会把一定数量的页面驻留在 TLB 中，而不致于被随机替换出去，这种机制有利于使操作系统提高性能，避免死锁。这种机制也使实时系统比较方便地为某一关键软件提供特定入口。

对每个页来说，JTLB 还维护该页面的 Cache 一致性属性，每个页都有特定的位来标记：不经过 Cache（Uncached），非一致性 Cache（Cacheable Noncoherent），或者是非 Cache 加速（Uncached Accelerated）。

3.1.2 指令 TLB

龙芯 2F 处理器的指令 TLB（ITLB）有 16 个表项，它最小化了 JTLB 的容量，并通

过一个大的相联阵列缩短了映射时的时间关键路径，降低了功率。每个 ITLB 表项只能映射一页，页面大小由 PageMask 寄存器来指定。ITLB 指令地址的映射和数据地址的映射能并行执行，从而提高了性能。当 ITLB 中的表项失效时，从 JTLB 中查找相应的表项，随机选择一个 ITLB 表项进行替换，ITLB 的操作对用户是完全透明的。处理器并没有保证 ITLB 与 JTLB 的一致，如果 JTLB 被修改时要求 ITLB 也要修改，则需要使用核心态指令刷新 ITLB，否则 ITLB 可能保持旧值。

3.1.3 命中和失效

如果虚拟地址与 TLB 中某个表项的虚拟地址一致（即 TLB 命中），则物理页面号就从 TLB 中取出，并和偏移连接组成物理地址。

如果虚拟地址与 TLB 中任何表项的虚拟地址都不一致（即 TLB 失效），则 CPU 产生一个异常，并由软件根据内存中存放的页表重新填写 TLB。软件既可以重写指定的 TLB 表项，也可以使用硬件提供的机制重写任意一个 TLB 表项。

3.1.4 多项命中

龙芯 2F 处理器对 TLB 中虚地址不只与一个表项的虚地址一致的情况，没有提供任何的探测和禁用机制，这一点不像早期的 MIPS 处理器的设计。多项命中并不会物理地破坏 TLB，因此多项命中的探测机制是不必要的。多项命中的情况没有被定义，因此软件要控制不要让多项命中的情况发生。

3.2 处理器模式

龙芯 2F 处理器有 3 种工作模式，但是与其它 MIPS 处理器不同，龙芯 2F 处理器只支持一种地址模式，一种指令集模式和一种尾端模式。

3.2.1 处理器工作模式

以下三种模式的处理器优先级依次降低：

- **内核模式**（最高的系统优先级）：在这种模式下处理器可以访问和改变任何寄存器，操作系统最内层的内核运行在内核模式；
- **管理模式**：处理器的优先级降低，操作系统的一些不太关键的部分运行在该模式；
- **用户模式**（最低的系统优先级）：该模式下使不同的用户间不致互相干扰。

三种模式的切换是由操作系统（在内核模式）置位状态寄存器 KSU 的相应位来实现的。当出现一个错误（ERL 位置位）或出现一个例外（EXL 位置位）时，处理器被强制切换到内核模式。表 3-1 列出了三种模式切换时 KSU、EXL 和 ERL 的置位情况，空的表项可以不必关心。

表 3-1 处理器的工作模式

KSU 4:3	ERL 2	EXL 1	描述
10	0	0	用户模式
01	0	0	管理模式
00	0	0	内核模式
	0	1	例外级别
	1		错误级别

3.2.2 地址模式

龙芯 2F 处理器只支持 64 位的虚拟地址模式，并且硬件保证兼容 32 位的地址模式。

3.2.3 指令集模式

龙芯 2F 处理器实现了完备的 MIPS III 指令集，另外还增加了一些整型和浮点指令，增加的指令见附件 A 和附录 B。

3.2.4 尾端模式

龙芯 2F 处理器只工作在小尾端模式。

3.3 地址空间

本节叙述的是虚拟地址空间，物理地址空间，和经过 TLB 进行虚实地址转换的方法。

3.3.1 虚拟地址空间

龙芯 2F 处理器有三个虚拟地址空间：用户地址空间、管理地址空间和内核地址空间，每个空间都是 64 位的，并且包含一些不连续的地址空间段，最大的段为 1T(2^{40})字节。

3.3.4 节到 3.3.6 节分别描述了这三种地址空间。

3.3.2 物理地址空间

通过使用 40 位地址，处理器的物理地址空间大小为 1T(2^{40})字节。以下小节将详述虚实地址转换的方法。

3.3.3 虚实地址转换

进行虚实地址转换时，首先比较处理器给出的虚拟地址和 TLB 中存放的虚拟地址。当虚页号（VPN）等于某个 TLB 表项的 VPN 域，并且如果下面两种情况中的任何一种成立：

- TLB 表项的 Global 位为 1
- 两个虚拟地址的 ASID 域一样。

TLB 就命中了。如果不满足以上条件，那么 CPU 会产生 TLB 失效异常，以使软件

能够根据内存中存放的页表重新填写 TLB。

如果 TLB 命中了，则物理页号将从 TLB 中取出，并与页内偏移量 Offset 合并，形成物理地址。页内偏移量 Offset 在虚实地址转换的过程中不经过 TLB。

（1）用虚页号（VPN）表示的虚拟地址（VA）与 TLB 中的对应域作比较；

（2）如果有一致的情况，则表示物理地址（PA）高位的页框号（PFN）从 TLB 中输出；

（3）偏移量 Offset 不经过 TLB，而是和 PFN 合并形成物理地址

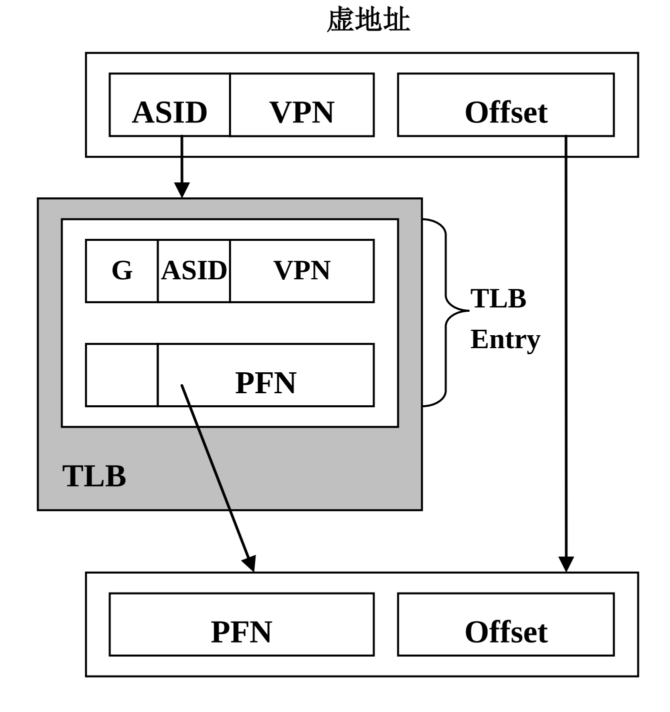


图 3-1 虚实地址转换概览

图 3-1 所示为虚实地址转换，虚拟地址被一个 8 位的地址空间标识符（ASID）扩展了，该措施降低了上下文切换时进行 TLB 刷新的频率。ASID 存放在 CP0 EntryHi 寄存器中。Global 位（G）在相应的 TLB 表项中。

图 3-2 显示了 64 位模式的虚实地址转换过程，这个图显示了最大页面 16MB 和最小页面 4KB 的情况。

图的上半部分显示了页面大小为 4K 字节的情况，页内偏移量 Offset 占用虚拟地址中的 12 位，虚拟地址中剩下的 28 位为虚页号 VPN，用于索引 4G 个页表表项；

图的下半部分显示了页面大小为 16M 字节的情况，页内偏移量 Offset 占用虚拟地址中的 24 位，虚拟地址中剩下的 16 位为虚页号 VPN，用于索引 64K 个页表表项。

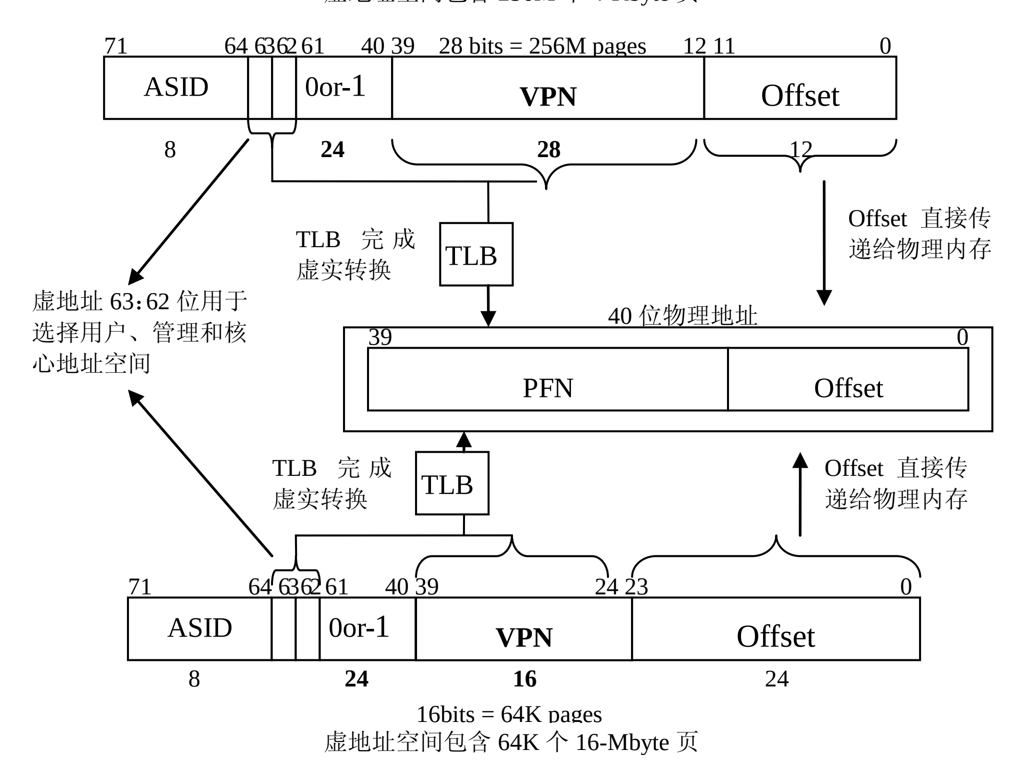


图 3-2 64 位模式虚拟地址转换

3.3.4 用户地址空间

在用户模式下，只有一个称为用户段（User Segment）的单独、统一的虚拟地址空间，其大小为 1T（2^{40}）字节，名字为 XUSEG。

图 3-3 显示了用户虚拟地址空间，可以在用户模式、管理模式、内核模式下访问。

用户段从地址 0 处开始，当前活动的用户进程驻留在该段（XUSEG）中。在不同模式下，TLB 对 XUSEG 段的映射处理方式都一样，并控制是否可以访问 Cache。

当处理器的 Status 寄存器的值同时满足三个条件：KSU=10_2、EXL=0、ERL=0 时，处理器工作在用户模式下。

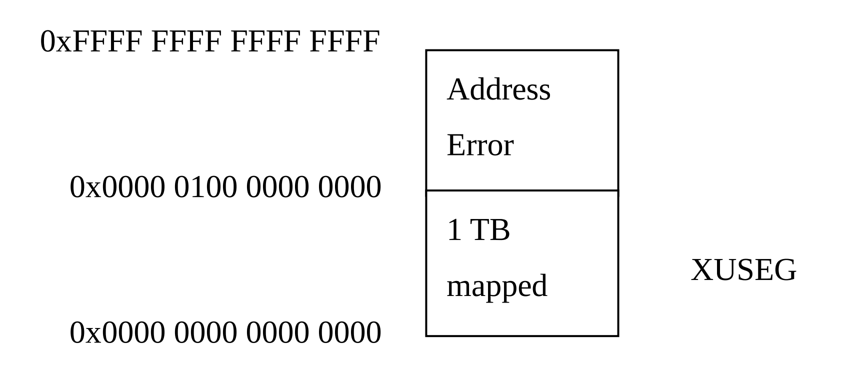


图 3-3 用户模式下用户虚拟地址空间概况

所有可用的用户模式下虚拟地址的第 63 位到第 40 位必须都为 0，访问任何一个第 63 位到第 40 位不全为 0 的地址都将导致地址错误异常，在 XUSEG 地址段的 TLB 缺失使用 XTLB 重填向量。龙芯 2F 处理器的 XTLB 重填向量与 32 位模式下 TLB 的重填向量有相同的例外入口地址。

3.3.5 管理地址空间

管理模式是为分层结构的操作系统设计的。在分层结构的操作系统中，真正的内核运行在内核模式下，操作系统的其余部分运行在管理模式下。管理地址空间提供了管理模式下程序访问的代码和数据空间。管理地址空间的 TLB 缺失由 XTLB 重填处理器来处理。

管理模式和内核模式都可访问管理地址空间。

当处理器的 Status 寄存器的值同时满足三个条件：KSU=01_2、EXL=0、ERL=0 时，处理器工作在管理模式下。图 3-4 显示了管理模式下的用户和管理地址空间概况。

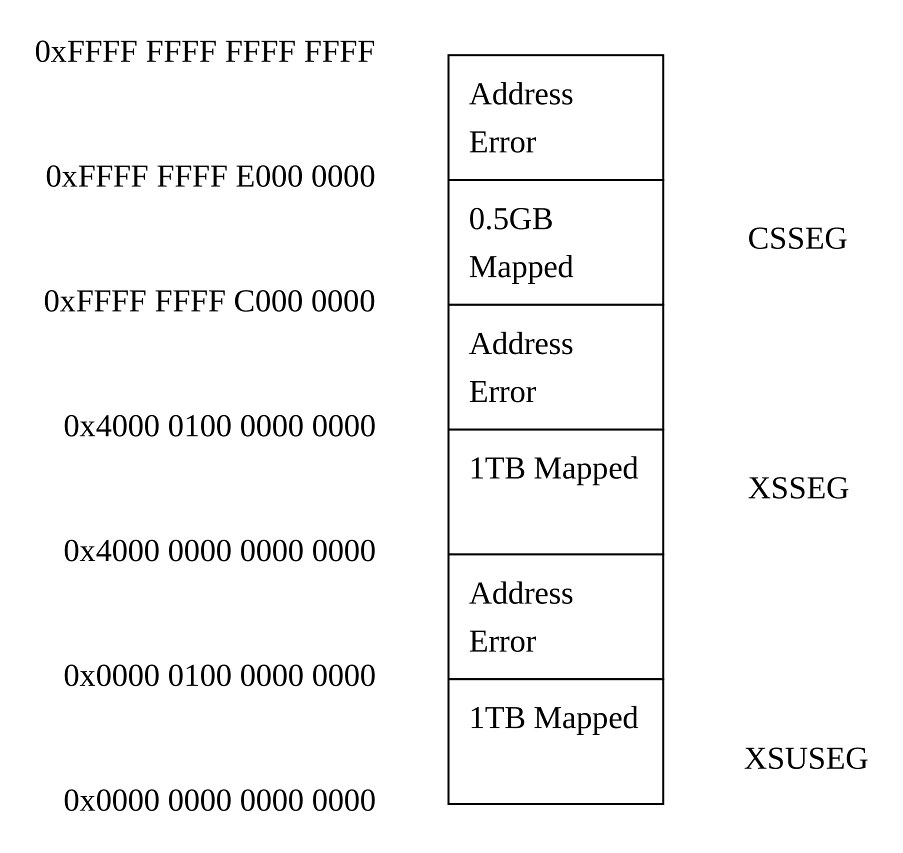


图 3-4 管理模式下用户空间和管理空间

- 64 位管理模式，用户地址空间（XSUSEG）

在管理模式下，当访问用户地址空间并且 64 位地址的最高两位（第 63 和第 62 位）为 00_2 时，程序使用一个名字为 XSUSEG 的虚拟地址空间，XSUSEG 覆盖了当前用户地址空间的全部 2^{40}（1T）字节。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。此地址空间从 0x0000 0000 0000 0000 开始，到 0x0000 00FF FFFF FFFF 结束。

- 64 位管理模式，当前管理地址空间（XSSEG）

在管理模式下，当 64 位地址的最高两位（第 63 和第 62 位）为 01_2 时，程序使用一个名字为 XSSEG 的当前管理虚拟地址空间。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。此地址空间从 0x4000 0000 0000 0000 开始，到 0x4000 00FF FFFF FFFF 结束。

- 64 位管理模式，独立管理地址空间（CSSEG）

在管理模式下，当 64 位地址的最高两位（第 63 和第 62 位）为 11_2 时，程序使用一个名字为 CSSEG 的独立管理虚拟地址空间。在 CSSEG 中的寻址与 32 位模式下在 SSEG 中的寻址是兼容的。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。此地址空间从 0xFFFF FFFF C000 0000 开始，到 0xFFFF FFFF DFFF FFFF 结束。

3.3.6 内核地址空间

当处理器的 Status 寄存器的值满足下述条件：KSU=00_2 或 EXL=1 或 ERL=1 时，处理器工作在内核模式下。

每当处理器检测到一个例外时便进入内核模式，并一直保持到执行例外返回指令（ERET）。ERET 指令将处理器恢复到例外发生前所在的模式。

根据虚拟地址高位的不同，内核模式虚拟地址空间被分为不同的区域，如

图 3-5 所示。

- 64 位内核模式，用户地址空间（XKUSEG）

在内核模式下，当访问用户空间并且 64 位虚拟地址的最高两位为 00_2 时，程序使用一个名字为 XKUSEG 的虚拟地址空间，XKUSEG 覆盖了当前用户地址空间。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

- 64 位内核模式，当前管理地址空间（XKSSEG）

在内核模式下，当访问管理空间并且 64 位地址的最高两位为 01_2 时，程序使用一个名字为 XKSSEG 的虚拟地址空间，XKSSEG 是当前管理虚拟地址空间。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

- 64 位内核模式，物理地址空间（XKPHY）

在内核模式下，当 64 位地址的最高两位为 10_2 时，程序使用一个名字为 XKPHY 的虚拟地址空间，XKPHY 是八个 2^{40} 字节的内核物理地址空间的集合。访问任何地址第 58 到第 40 位不为 0 的存储单元都将引起地址错误。对 XKPHY 的访问不经过 TLB 进行地址变换，而是将虚拟地址的第 39 到第 0 位作为物理地址。虚拟地址的第 61 到第 59 位控制是否通过 Cache 和 Cache 的一致性属性，与表 3-2 描述的 TLB 页的 C 位值含义相同。

- 64 位内核模式，内核地址空间（XKSEG）

在内核模式下，当 64 位地址的最高两位为 11_2 时，程序使用以下两个地址空间之一：

- 内核虚拟地址空间 XKSEG，此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址；
- 四个 32 位内核兼容地址空间，下一小节详述。

● 64 位内核模式，兼容地址空间（CKSEG1：0，CKSSEG，CKSEG3）

在内核模式下，64 位地址的最高两位为 11_2，并且虚拟地址的第 61 到第 31 位所有位都等于 1 时，程序使用的以下四个 512M 字节地址空间中的一个，具体哪一个根据第 30、29 位决定：

地址	区域	段
0xFFFF FFFF FFFF FFFF		
	0.5GB Mapped	CKSEG3
0xFFFF FFFF E000 0000		
	0.5GB Mapped	CKSSEG
0xFFFF FFFF C000 0000		
	0.5GB Unmapped Cached	CKSEG1
0xFFFF FFFF A000 0000		
	0.5GB Unmapped Cached	CKSEG0
0xFFFF FFFF 8000 0000		
	Address Error	
0xC000 00FF 8000 0000		
	Mapped	XKSEG
0xC000 0000 0000 0000		
	Unmapped	XKPHY
0x8000 0000 0000 0000		
	Address Error	
0x4000 0100 0000 0000		
	1TB Mapped	XKSSEG
0x4000 0000 0000 0000		
	Address Error	
0x0000 0100 0000 0000		
	1TB Mapped	XKUSEG
0x0000 0000 0000 0000		

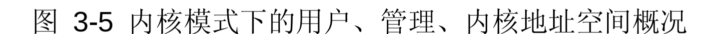


图 3-5 内核模式下的用户、管理、内核地址空间概况

- CKSEG0：该 64 位虚拟地址空间不经过 TLB，与 32 位模式下的 KSEG0 兼容。Config 寄存器的 K0 域控制是否通过 Cache 和 Cache 的一致性属性，
- CKSEG1：该 64 位虚拟地址空间不经过 TLB 也不经过 Cache，与 32 位模式下的 KSEG1 兼容。
- CKSSEG：该 64 位虚拟地址空间为当前管理虚拟地址空间，与 32 位模式下的 KSSEG 兼容。
- CKSEG3：该 64 位虚拟地址空间为内核虚拟地址空间，与 32 位模式下的 KSEG3 兼容。

3.4 系统控制协处理器

系统控制协处理器(CP0)负责支持存储管理，虚实地址转换，例外处理，以及一些特权操作。龙芯 2F 处理器有 26 个 CP0 寄存器和一个 64 项的 TLB，每个寄存器都有唯一的寄存器号。下面的章节将给出与内存管理相关的寄存器的概述。

3.4.1 TLB 表项的格式

图 3-6 表示 TLB 表项的格式，项中的每个域在 EntryHi，EntryLo0，EntryLo1，PageMask 寄存器中都有相应的域。

EntryHi，EntryLo0，EntryLo1，以及 PageMask 寄存器和 TLB 项的格式类似。唯一的不同就是 TLB 项有一个 Global 域（G 位），EntryHi 寄存器中没有，而作为保留域出现。图 3-7、图 3-8 和图 3-9 分别表示了在图 3-6 TLB 项的各个域。

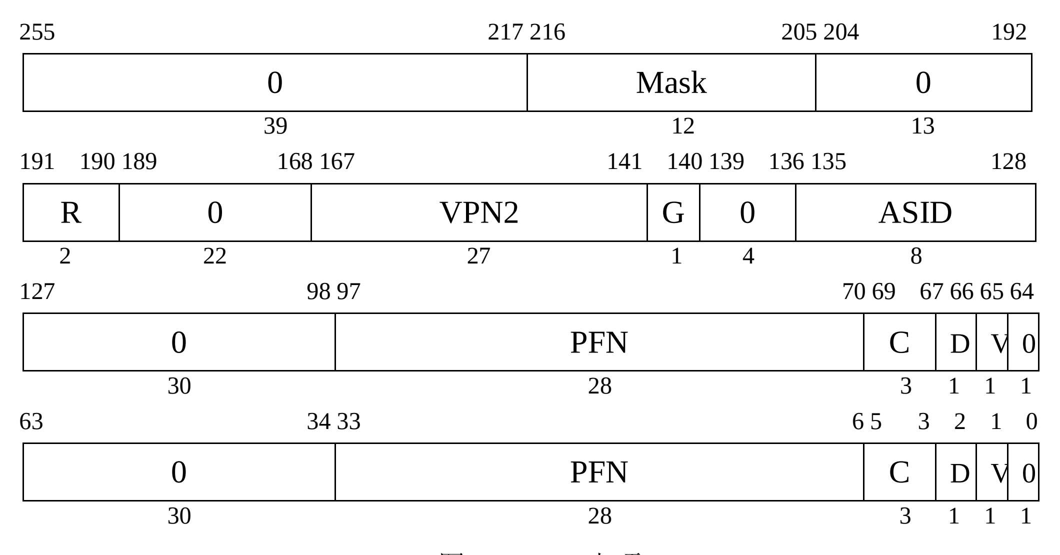
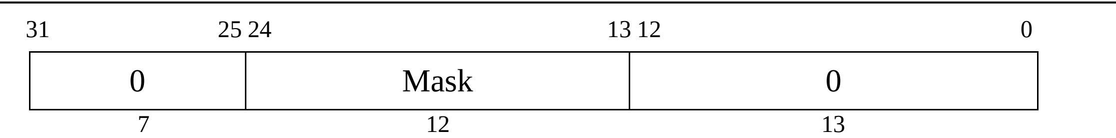


图 3-6 TLB 表项



Mask……页比较掩码

0………..保留。写入必须是 0，读时返回 0。

图 3-7 PageMask 寄存器

63 62 61	40 39	13 12	8 7	0
R	Fill	VPN2	0	ASID
2	22	27	5	8

VPN2….虚页号除 2（映射 2 个页）。

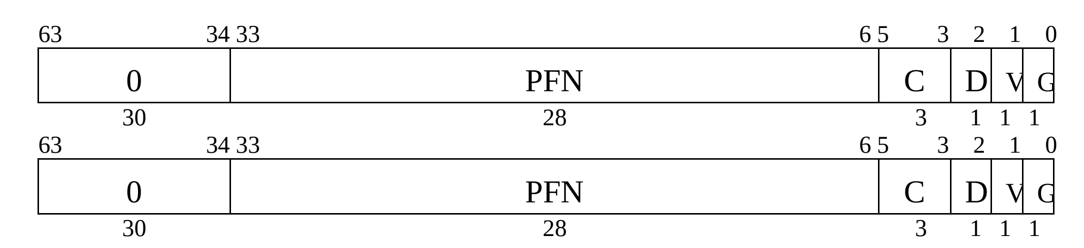
ASID….地址空间 ID 域。一个 8 位域，用于让多个进程共享 TLB；对于与其它进程同样的虚页号，每个进程有不同的映射。

R………区域。（00->用户，01->管理，11->核心）用户匹配虚地址 63:62 位。

Fill…….保留。读为 0，写忽略。

0………保留。写入必须是 0，读时返回 0。

图 3-8 EntryHi 寄存器



PFN…页框号；物理地址的高位。

C…….指定 TLB 页的一致性属性；见表 3-2。

D…….脏位。如该位值设置为 1，则对应的页标记为脏，因而可写。该位实际上是写保护位，软件可用此位保护数据的改动。

V…….有效位。该位设置表示 TLB 表项是有效的；否则，将产生 TLBL/TLBS 失效。

G…….全局位。如果 Lo0 和 Lo1 中对应的位都设置为 1，则在 TLB 查找时处理器忽略 ASID。

0……..保留。写入必须是 0，读时返回 0。

图 3-9 EntryLo0 和 EntryLo1 寄存器

TLB 页一致性属性位（C）指定访问该页时是否需要通过 Cache，如果通过 Cache，则需要选择 Cache 的一致性属性。表 3-2 表示 C 位对应的 Cache 一致性属性。

表 3-2 TLB 页的 C 位的值

C(5:3) 值	Cache一致性属性
0	保留
1	保留
2	非高速缓存（Uncached）

3	非一致性高速缓存（Cacheable Noncoherent）
4	保留
5	保留
6	保留
7	非高速缓存加速（Uncached Accelerated）

3.4.2 CP0 寄存器

表 3-3 列出了与内存管理相关的 CP0 寄存器，第 5 章对 CP0 寄存器进行了完备的描述。

表 3-3 内存管理相关的 CP0 寄存器

寄存器号	寄存器名
0	Index
1	Random
2	EntryLo0
3	EntryLo1
5	PageMask
6	Wired
10	EntryHi
15	PRID
16	Config
17	LLAddr
28	TagLo
29	TagHi

3.4.3 虚拟地址到物理地址的转换过程

在虚地址到物理地址转换时，CPU 将虚地址的 8 位 ASID（如果全局位 G 没有设置）和 TLB 项的 ASID 进行比较，看是否匹配。在比较 ASID 的同时还需要根据页掩码（PageMask）的值将虚地址的高 15~27 位和 TLB 项的虚页号进行匹配比较。如果有 TLB 项匹配，从匹配的 TLB 项中取出物理地址和访问控制位（C，D 和 V）。对一个有效的地址转换来说，匹配的 TLB 项的 V 位必须设置，但是在匹配比较时不考虑 V 位的值。

图 3-10 图示了 TLB 地址转换过程。

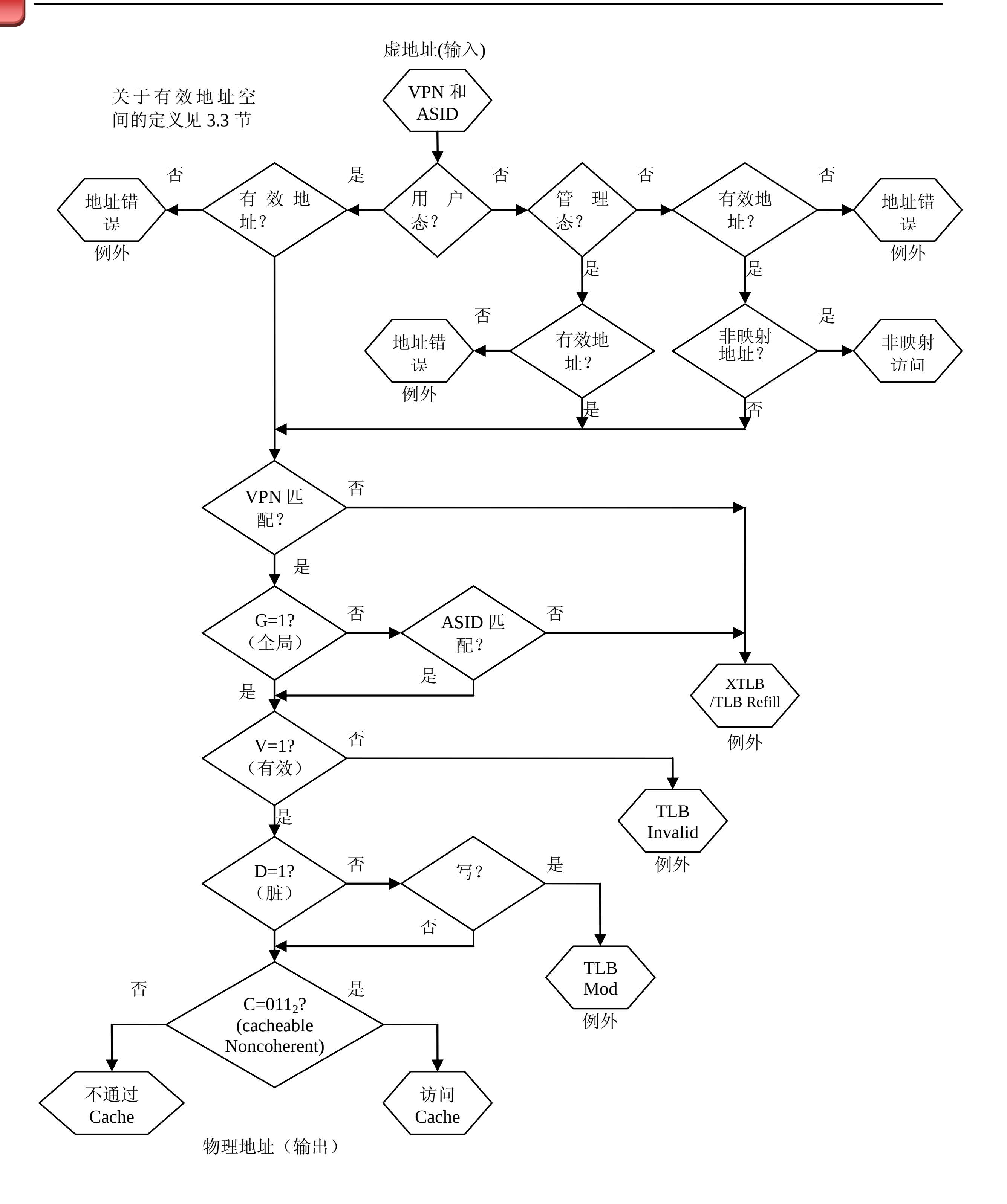


图 3-10 TLB 地址转换

3.4.4 TLB 失效

如果没有任何一 TLB 项匹配虚地址，引发一个 TLB 不命中例外。如果访问控制位(D 和 V)指示访问不是合法的，引发一个 TLB 修改或者 TLB 无效例外。如果 C 位等于 011_2，被检索到的物理地址通过 Cache 访问内存， 否则不通过 Cache。

3.4.5 TLB 指令

表 3-4 列出了所有的 CPU 所提供的用于和 TLB 操作有关的指令。

表 3-4 TLB 指令

操作码	指令描述
TLBP	在 TLB 中搜索匹配项
TLBR	读索引的 TLB 项
TLBWI	写索引的 TLB 项
TLBWR	写随机的 TLB 项

3.4.6 代码例子

第一个例子是如何配置 TLB 表项来映射一对 4KB 的页面。实时系统的内核大多都这么做，这种简单的内核 MMU 只用于进行内存保护，所以静态映射就足够了，在所有静态映射的系统中所有 TLB 例外都被作为是错误条件（不可访问）。

```
mtc0 r0,C0_WIRED              # make all entries available to random replacement
li r2, (vpn2<<13)|(asid & 0xff);
mtc0 r2, C0_ENHI              # set the virtual address
li r2, (epfn<<6)|(coherency<<3)|(Dirty<<2)|Valid<<1|Global)
mtc0 r2, C0_ENLO0             # set the physical address for the even page
li r2, (opfn<<6)|(coherency<<3)|(Dirty<<2)|Valid<<1|Global)
mtc0 r2, C0_ENLO1             # set the physical address for the odd page
li r2, 0                      # set the page size to 4KB
mtc0 r2,C0_PAGEMASK
li r2, index_of_some_entry    # needed for tlbwi only
mtc0 r2, C0_INDEX             # needed for tlbwi only
tlbwr                         # or tlbwi
```

一个完备的虚拟存储操作系统（如 UNIX），用 MMU 进行内存保护，并进行主存和大容量存储设备的换页。这个机制使程序可以访问更大的存储设备而不仅仅局限于系统物理分配的空间。这个依赖于请求调页的机制需要动态页面映射。动态映射通过一系列不同类型的 MMU 例外实施，TLB 重填是这种系统中最常见的例外。下面是一个可能的 TLB 重填例外控制。

```
refill_exception:
    mfc0 k0,C0_CONTEXT
    sra k0,k0,1               # index into the page table
    lw k1,0(k0)               # read page table
    lw k0,4(k0)
    sll k1,k1,6
    srl k1,k1,6
```

```
mtc0 k1,C0_TLBLO0
sll k0,k0,6
srl k0,k0,6
mtc0 k0,C0_TLBLO1
tlbwr                    # write a random entry
eret
```

这个例外控制处理非常简单，因为它的经常执行会影响系统性能，这就是 TLB 重填例外分配独立的例外向量的原因。这段代码假设需要的映射在主存储器的页表中已经建立起来了。如果没有建立起来，那么在 ERET 指令后将发生 TLB 失效例外。TLB 失效例外很少发生，这是有益的，因为它必须计算期望的映射，并可能需要从后援存储器中读取部分页表。TLB 修改例外用于实现只读页面和标记进程清除代码需要修改的页。为了保护不同的进程和用户不受相互的干扰，虚拟存储操作系统通常在用户模式执行用户程序。下面的例子表示如何从内核模式进入用户模式。

```
mtc0 r10, C0_EPC                           # assume r10 holds desired usermode address
mfc0 r1, C0_SR                             # get current value of Status register
and r1,r1, ~(SR_KSU || SR_ERL)             # clear KSU and ERL field
or r1, r1, (KSU_USERMODE || SR_EXL)        # set usermode and EXL bit
mtc0 r1, C0_SR
eret                                       # jump to user mode
```

4 Cache 的组织和操作

龙芯 2F 使用了三个独立的 Cache：

一级指令 Cache：64KB 的容量，采用四路组相联的结构。

一级数据 Cache：64KB 的容量，采用四路组相联的结构，采用写回策略。

二级混合 Cache：片上 Cache，512KB 的容量，采用四路组相联的结构，采用写回策略。

4.1 Cache 概述

访问一次一级 Cache 需要 4 个时钟周期。每个一级 Cache 都有它们自己的数据通路，从而可以同时访问两个 Cache。其中，指令 Cache 的读通路是 128 位，回填通路是 64 位；而数据 Cache 的读写和回填数据通路都是 64 位。

二级 Cache 使用的是 256 位的数据通路，它只有在一级 Cache 失效时才被访问。二级 Cache 和一级 Cache 不能同时访问，当一级 Cache 失效时，访问二级 Cache 至少会增加 11 个周期的失效代价。二级 Cache 以每个时钟周期 64 位数据的速度对一级 Cache 进行回填。

一级 Cache 采用虚地址索引和物理地址标志，而二级 Cache 的索引和标志采用的都是物理地址。虚地址索引可能会引起不一致问题，目前龙芯 2F 使用操作系统来解决，将来可以通过硬件方式来解决。

多级 Cache 的结构给一级 Cache 的刷新操作带来一些新的问题。没有使用二级 Cache 时，一级 Cache 的刷新只需考虑把数据更新到主存就可以了。然而增加了二级 Cache 后，要先把数据更新到二级 Cache 中，然后必须刷新二级 Cache，再把数据更新到主存中。由于一级 Cache 和二级 Cache 保持包含关系，也可以只执行刷新二级 Cache 的指令，完成将二级 Cache 包含的一级 Cache 块从一级 Cache 更新到二级 Cache，再从二级 Cache 更新到主存的操作。

4.1.1 非阻塞 Cache

龙芯 2F 实现了非阻塞 Cache 技术。非阻塞 Cache 是通过允许 Cache 失效访存操作后面的多个 Cache 失效或命中的访存操作继续进行，来提高系统的整体性能。

在一个阻塞 Cache 的设计中，当发生某个 Cache 失效时，处理器将暂停后续的访存操作。此时，处理器开始一个存储周期，取出被请求的数据，将其填入 Cache 中，然后再恢复执行。这个操作过程会占用较多的时钟周期，具体多少取决于存储器系统的设计。

然而在非阻塞 Cache 设计中，Cache 并不会在某个失效上暂停。龙芯 2F 支持多重失效下的命中，它最多可以支持 24 次 Cache 失效，这与 CP0 队列大小有关。

当一级 Cache 失效时，处理器会检查二级 Cache，看所需数据是否在其中，若二级

Cache 仍然失效，则需要访问主存储器。

龙芯 2F 中的非阻塞 Cache 结构能更有效的使用循环展开和软件流水。为了尽可能最大限度地发挥 Cache 的优势，在使用访存数据的指令之前，尽可能早的执行相应的 Load 操作。

针对那些需要顺序存取的 I/O 系统，龙芯 2F 的默认设置是采用阻塞式的 Uncached 访问方式。

4.1.2 替换策略

一级 Cache 和二级 Cache 均采用随机替换算法。

4.1.3 Cache 的参数

表 4-1 给出了三个 Cache 的一些参数

表 4-1 Cache 参数

参数	指令 Cache	数据 Cache	二级 Cache
Cache 大小	64KB	64KB	512KB
相联度	4 路组相联	4 路组相联	4 路组相联
替换策略	随机法	随机法	随机法
块大小(line size)	32 字节	32 字节	32 字节
索引(Index)	虚地址 13:5 位	虚地址 13:5 位	物理地址 16:5 位
标志(Tag)	物理地址 39:12 位	物理地址 39:12 位	物理地址 39:12 位
写策略	不可写	写回法	写回法
读策略	非阻塞（2 个同时）	非阻塞(24 个同时)	非阻塞（8 个同时）
读顺序	关键字优先	关键字优先	关键字优先
写顺序	不可写	顺序式	顺序式

4.2 一级指令 Cache

一级指令 Cache 大小是 64KB，采用的是四路组相联结构。Cache 块大小（通常也被称作 Cache 行）为 32 字节，可以存放 8 条指令。由于龙芯 2F 采用 128 位的读通路，所以每个时钟周期可以取四条指令送到超标量调度单元。

4.2.1 指令 Cache 的组织

图 4-1 给出了一级指令 Cache 的组织结构。该 Cache 采用四路组相联的映射方式，其中每组包括 512 个索引项。根据索引(Index)选择相应的标志(Tag)和数据(Data)。从 Cache 读出 Tag 后，它被用来和虚地址中的被转换的部分进行比较，从而确定包含正确数据的组。

当一级指令 Cache 被索引时，四个组都会返回它们相应的 Cache 行，Cache 行大小为 32 字节，Cache 行采用了 28 位作为标志和 1 位作为有效位。图 4-2 描述了指令 Cache 行

格式。

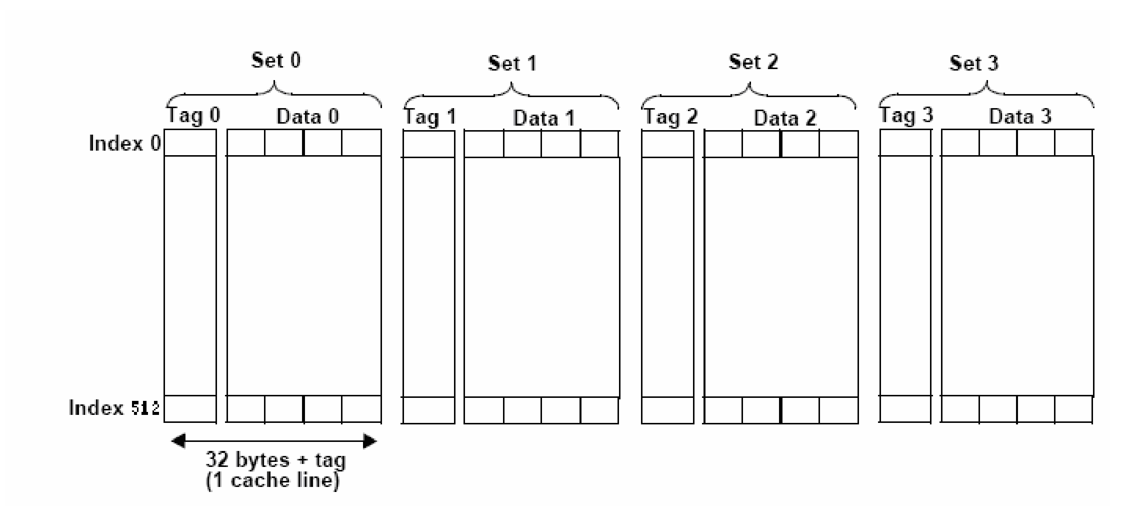
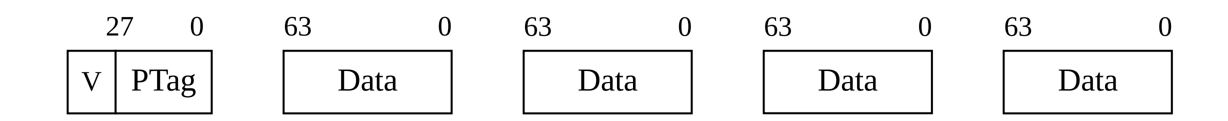


图 4-1 指令 Cache 的组织



V……….Tag 有效位
PTag……28 位的物理地址 Tag（物理地址的位 39:12）
Data…... Cache Data

图 4-2 指令 Cache 行格式

4.2.2 指令 Cache 的访问

龙芯 2F 指令 Cache 采用虚地址索引和物理地址标志的四路组相联结构。图 4-3 给出了访问一次指令 Cache 时，虚地址如何被分解。

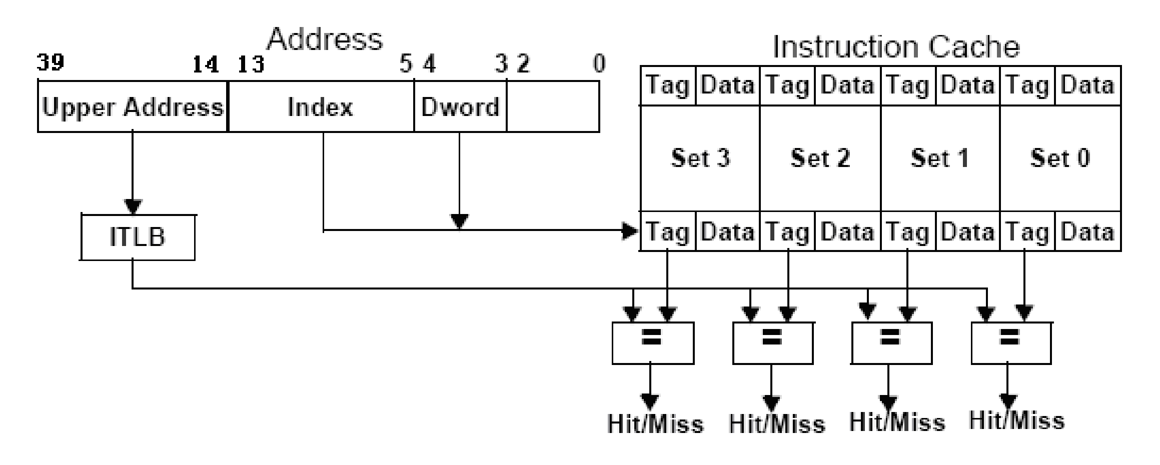


图 4-3 指令 Cache 访问

如图 4-3 所示，地址的低 14 位被用作指令 Cache 的索引。其中 13:5 位用于索引 512 个项。其中每个项中又包含四个 64 位的双字，使用 4:3 位在这四个双字中进行选择。

当对 Cache 索引时，从 Cache 中取出四个块中的 Data 和相应的物理地址 Tag，同时，高位地址通过指令 TLB（Instruction Translation Look-aside Buffer，简称 ITLB）进行转换，将转换后的地址与取出的四个组中的 Tags 进行比较，若存在一个 Tag 与其匹配，则使用该组中的数据。这就被称为一次“一级 Cache 命中（Hit）”。若四组的 Tag 都不与其匹配，那么中止操作，并开始访问二级 Cache。这就被称为“一级 Cache 失效（miss）”。

4.3 一级数据 Cache

数据 Cache 的容量为 64KB，采用四路组相联的结构。Cache 块大小为 32 字节，即可以存放 8 个字。数据 Cache 的读写数据通路都是 64 位。

数据 Cache 使用的是虚地址索引，物理地址标志。操作系统可以解决可能由虚地址引起的一致性问题。数据 Cache 是非阻塞的，也就意味着，数据 Cache 中的一次失效不会引起流水线的停顿。

数据 Cache 采用的写策略是写回法，即写数据到一级 Cache 的操作不会引起二级 Cache 和主存的更新。写回策略减少了一级 Cache 到二级 Cache 的通信量，从而提高了全局性能。只有在数据 Cache 行被替换出去时，数据才会被写到二级 Cache 中。

4.3.1 数据 Cache 的组织

图 4-4 给出了数据 Cache 的组织结构。这是一个四路组相联的 Cache，其中含有 512 个索引项。当对 Cache 索引时，同时访问四个组中的 Tag 和 Data。然后将四个组中的 Tag 与转换后的物理地址部分进行比较，从而确定命中哪一个数据行。

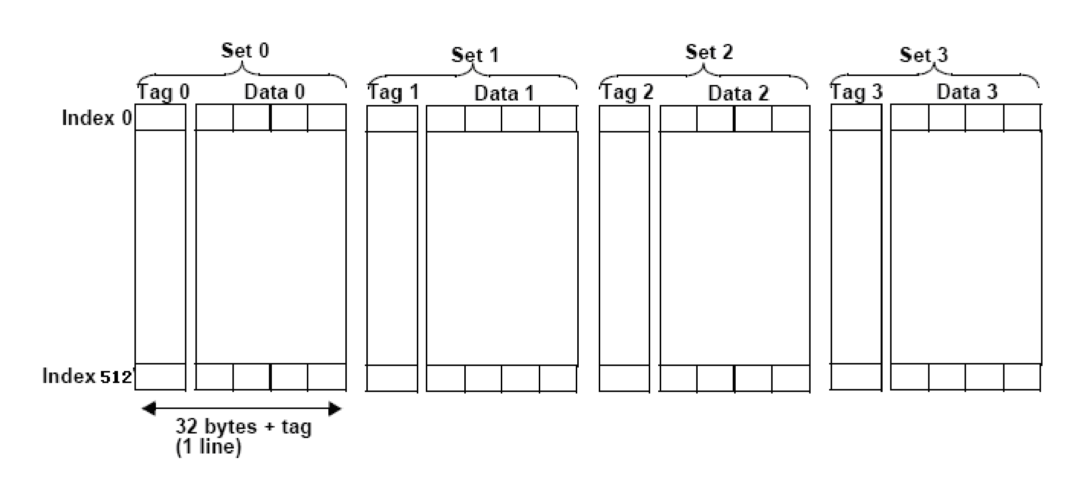
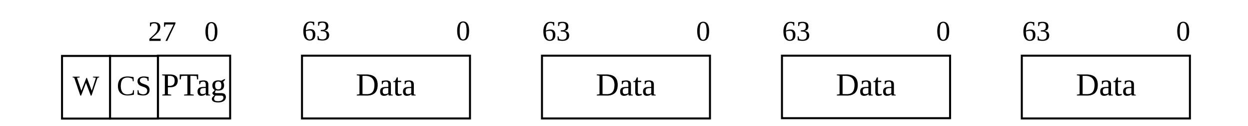


图 4-4 数据 Cache 的组织结构

当索引数据 Cache 时，四个组中都会返回它们各自相应的 Cache 行。Cache 块大小为 32 字节，Cache 行使用了 28 位作为物理标志地址 1 位脏位和 2 位状态位。图 4-5 给出了一个数据 Cache 行的格式。



W……….写回位（Cache 行写回后设置）
CS………主 Cache 状态
00_2 = Invalid（无效）
01_2 = Shared（共享）
10_2 = Exclusive（独占）
11_2 = Dirty（脏）
PTag……28 位的物理地址 Tag（物理地址的位 39:12）
Data…... Cache Data

图 4-5 数据 Cache 行格式

4.3.2 数据 Cache 的访问

龙芯 2F 数据 Cache 采用虚地址索引和物理地址标志的四路组相联结构。图 4-6 给出了访问一次数据 Cache 时，虚地址如何被分解。

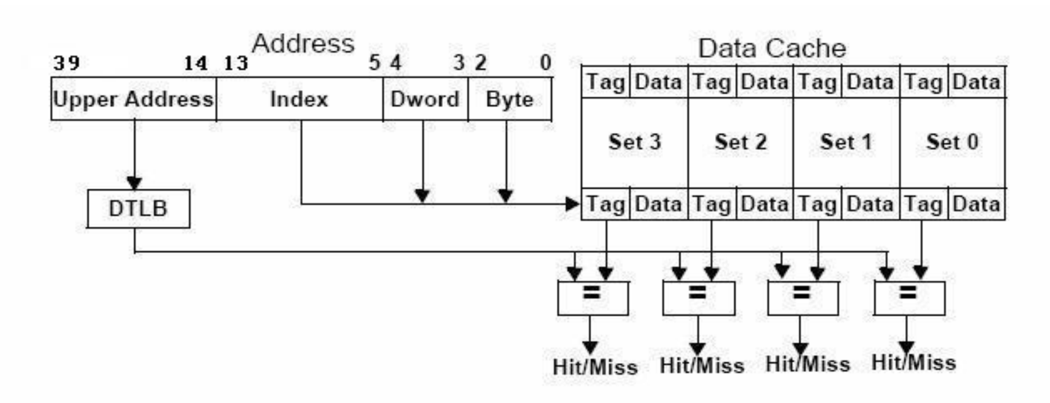


图 4-6 数据 Cache 访问

如图 4-6 所示，地址的低 14 位用作对数据 Cache 的索引。其中 13:5 位用作索引 512 个项，其中每个项又包括 4 个 64 位的双字。使用 4:3 位对四个双字进行选择，2:0 位用作选择一个双字的八个字节中的某一个字节。

4.3.3 数据 Cache 失效的处理

数据 Cache 访问失效的取数指令，访问二级 Cache。如果二级 Cache 命中，则将从二级 Cache 取回的 Cache 块送回一级 Cache。如果二级 Cache 失效，则访问内存，用从内存取回的值填充二级 Cache 和数据 Cache。

数据 Cache 访问失效的存数指令的处理采用存填充缓冲区（Store Fill Buffer）优化策略。访存队列 CP0 中提交后的 Cache 失效的存数操作不再堵在访存队列中等待 Cache 块的填充。如果 Cache 不命中，在该存数操作提交后把相应的写操作送到访存失效队列后

立即退出 CP0 队列以防止 CP0 队列堵塞。失效的存数指令项访问二级 Cache，如果二级 Cache 命中，则把存数操作的值和从二级 Cache 取回的 Cache 块进行合并送回数据 Cache。如果二级 Cache 访问不命中，则在访存失效队列中等待收集为全修改 Cache 块。如果收集为全修改 Cache 块，直接填充数据 Cache。如果等待收集过程中发生取数指令访问对应 Cache 块、访存失效队列满或处理器执行同步指令、Cache 指令需要清空访存失效队列等操作，则将未收集满的存数操作的值和从内存取回的 Cache 块的值进行合并送回数据 Cache。该方法实现了 Store Fill Buffer 的功能，而且不需要设计独立的存数指令收集缓冲区，避免了增加额外的硬件开销，又避免了存数指令收集缓冲区与访存失效队列互相查询以保证数据一致性的开销。通过 Store Fill Buffer 的优化，有效地提高了处理器的带宽利用率。

4.4 二级 Cache

龙芯 2F 采用一个片上的，四路组相联的二级 Cache。它的容量为 512KB，块大小为 32 字节。

二级 Cache 采用的写策略是写回法。写回策略减少了总线的通信量，从而提高了系统的全局性能。只有在二级 Cache 行被替换出去时，数据才会被写到内存中。

4.4.1 二级 Cache 的组织

二级 Cache 是混合 Cache，其中既包括指令也包括数据。

当索引二级 Cache 时，同时访问四组的 Data 和 Tag。将取出的四个 Tag 分别和访问的物理地址高位部分进行比较，来确定数据是否还驻留在 Cache 中。

当索引二级 Cache 时，四个组都会返回它们各自相应的 Cache 行。每个 Cache 行包含一个 32 字节的数据，23 位的物理地址标志和 2 位状态位。

4.4.2 二级 Cache 的访问

只有在一级 Cache 失效的情况下，才访问二级 Cache。二级 Cache 采用的是物理地址索引物理地址标志。图 4-7 给出了二级 Cache 访问的过程。

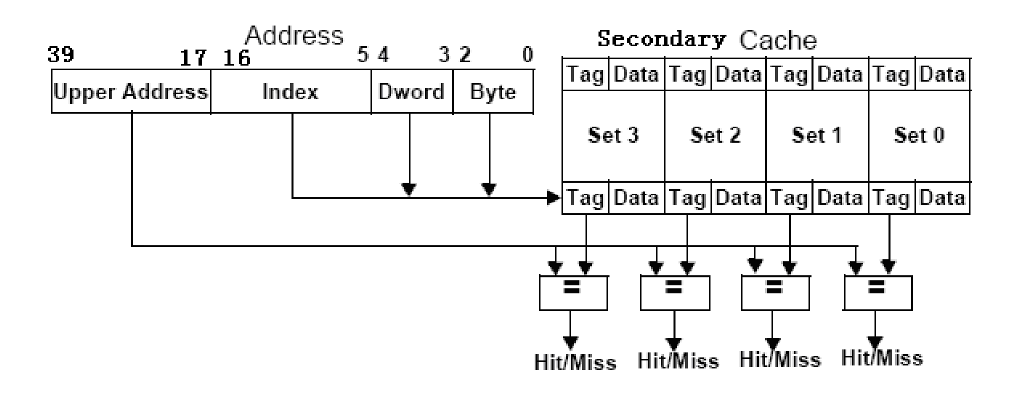


图 4-7 二级 Cache 访问

如图 4-7 示，低位地址用来索引二级 Cache。四个组中都会返回它们各自相应的 Cache 行。16:5 位被用作二级 Cache 的索引。每个被索引项都含有 4 个 64 位的双字数据。使用 4:3 位在 4 个双字中进行选择。2:0 位用于选择一个双字中的某 8 个字节。

4.5 Cache 算法和 Cache 一致性属性

龙芯 2F 实现表 4-2 所示的 Cache 算法和 Cache 一致性属性。

表 4-2 龙芯 2FCache 的一致性属性

属性分类	一致性代码
保留	0
保留	1
非高速缓存（Uncached）	2
非一致性高速缓存（Cacheable Noncoherent）	3
保留	4
保留	5
保留	6
非高速缓存加速（Uncached Accelerated）	7

4.5.1 非高速缓存(Uncached，一致性代码 2)

如果某个页采用非高速缓存算法时，那么对于在该页中任何位置的 Load 或 Store 操作，处理器都直接发射一个双字，部分双字，字，部分字的读或写请求给主存，而不通过任何一级 Cache。非高速缓存算法采用阻塞的方式实现。

4.5.2 非一致性高速缓存(Cacheable Noncoherent，一致性代码 3)

一个具有该属性的行可以驻留在 Cache 中，相应的存数和取数操作都只访问一级 Cache。当一级 Cache 失效时，处理器会检查二级 Cache，看是否有包含所请求的地址。如果二级 Cache 命中，则从二级 Cache 中填充数据。如果二级 Cache 不命中，则从主存中取出数据，并将其写入二级 Cache 和一级 Cache。

龙芯 2F 的 Cache 写策略采用写回法，因此只有当 Cache 行发生替换或软件执行 Cache 操作主动将 Cache 行的内容写回时，被修改的 Cache 行内容才写回下一级 Cache 或主存中。

由于系统中存在多个主设备可以访问主存，因此需要使用一种机制来保证 Cache 和主存中的数据一致性，这种机制被称为 Cache 一致性协议。而非一致性高速缓存机制（Cacheable Noncoherent）是指处理器没有提供硬件机制来解决 Cache 一致性的问题，需要通过软件采用 Cache 指令来主动维护 Cache 的一致性。

4.5.3 非高速缓存加速 (Uncached Accelerated，一致性代码 7)

非高速缓存加速属性用于优化在一个连续的地址空间中完成的一系列顺序的同一类型的 Uncached 存数操作。该优化方法是通过设置缓冲区来收集这种属性的存数操作。只要缓冲区不满，就可以把这些存数操作的数据存入缓冲区中。缓冲区和一个 Cache 行一样大小。把数据存储到缓冲区中就和存储到 Cache 中一样。当缓冲区满的时候，开始进行块写。在顺序存数指令的收集过程中，若有其他类型 Uncached 存数指令插入，则收集工作中止，缓冲区中保存的数据按字节写方式输出。

非高速缓存加速属性可以加速顺序的 Uncached 访问，它适用于对显示设备存储的快速输出访问。

4.6 Cache 的维护

在片上多级存储器结构中，必须要保证在进程切换前所有修改的数据已经更新到外部存储器。为了刷新片上写缓冲区，软件上使用 SYNC 指令。这条指令在所有悬挂的存数操作已经到达引脚外部总线前，和在所有悬挂的取数操作已经完成写相应目的寄存器前，将一直停顿处理器。

可以通过使用 Cache 指令来维护 Cache。龙芯 2F 在一级数据 Cache 和二级 Cache 中使用了两条“Hit”型 Cache 操作： Hit_Invalidate 和 Hit_Writeback_Invalidate。前者将相关的 Cache 行内容无效，一般在处理器读取 DMA 操作刚刚完成的设备输入缓存区内容前使用；后者将相关的 Cache 行内容写回下一级存储后再设置为无效，一般在处理器写即将开始 DMA 操作的设备输出缓冲区后使用。

5 CP0 控制寄存器

本章描述协处理器 0（Coprocessor 0，简称 CP0）的操作，主要内容包括 CP0 的寄存器定义以及龙芯 2F 处理器实现的 CP0 指令。CP0 寄存器用于控制处理器的状态改变并报告处理器的当前状态。这些寄存器通过 MFC0/DMFC0 指令来读或 MTC0/ DMTC0 指令来写。CP0 寄存器如表 5-1 所示。

当处理器运行在核心模式时或状态寄存器（Status 寄存器）中的第 28 位（CU0）被设置时，可以使用 CP0 指令。否则，执行 CP0 指令将产生“协处理器不可用例外”。

表 5-1 CP0 寄存器

寄存器号	寄存器名字	描述
0	Index	可写的寄存器，用于指定需要读/写的 TLB 表项
1	Random	用于 TLB 替换的伪随机计数器
2	EntryLo0	TLB 表项低半部分中对应于偶虚页的内容（主要是物理页号）
3	EntryLo1	TLB 表项低半部分中对应于奇虚页的内容（主要是物理页号）
4	Context	32 位寻址模式下指向内核的虚拟页转换表（PTE）
5	Page Mask	设置 TLB 页大小的掩码值
6	Wired	固定连线的 TLB 表项数目（指不用于随机替换的低端 TLB 表项）
7		保留
8	BadVaddr	错误的虚地址
9	Count	计数器
10	EntryHi	TLB 表项的高半部分内容（虚页号和 ASID）
11	Compare	计数器比较
12	Status	处理器状态寄存器
13	Cause	最近一次例外的原因
14	EPC	例外程序计数器
15	PRID	处理器修订版本标识号
16	Config	配置寄存器（Cache 大小等）
17	LLAddr	链接读内存地址
18	Watch	虚地址空间访问陷阱地址
19		保留
20	Xcontext	64 位寻址模式下指向内核的虚拟页转换表（PTE）
21		保留

寄存器号	寄存器名字	描述
22	Diagnose	使能/禁用 BTB,RAS 以及清空 ITLB 表
23		保留
24	PCLo	性能计数器的低半部分
25	PCHi	性能计数器的高半部分
26		保留
27		保留
28	TagLo	CACHE TAG 寄存器的低半部分
29	TagHi	CACHE TAG 寄存器的高半部分
30	ErrorEPC	错误例外程序计数器
31		保留

5.1 Index 寄存器(0)

Index 寄存器是个 32 位可读/写的寄存器，其中最后六位的值用于索引 TLB 的表项。寄存器的最高位表示 TLB 探测(TLBP)指令执行是否成功。

Index 寄存器的值指示 TLB 读(TLBR)和 TLB 索引写(TLBWI)指令操作的 TLB 表项。图 5-1 表示 Index 寄存器的格式，表 5-2 描述了 Index 寄存器各域的含义。

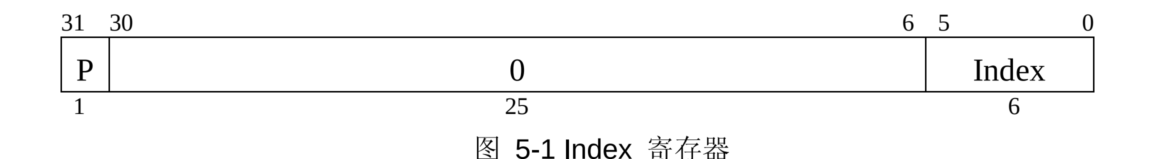


图 5-1 Index 寄存器

表 5-2 Index 寄存器各域描述

域	描述
P	探测失败。上一次 TLB 探测指令（TLBP）没有成功时置 1
Index	指示 TLB 读指令和 TLB 索引写指令操作的 TLB 表项的索引值
0	保留。必须按 0 写入，读时返回 0。

5.2 Random 寄存器(1)

Random 寄存器是个只读寄存器，其中低六位索引 TLB 的表项。每执行完一条指令，该寄存器值减 1。同时，寄存器值在一个上界和一个下界之间浮动，上下界具体是：

- 下界等于保留给操作系统专用的 TLB 项数（即 Wired 寄存器的内容）。
- 上界等于整个 TLB 的项数减 1（最大为 64-1）。

Random 寄存器指示将由 TLB 随机写指令操作的 TLB 项。从这个目的来说，无需读此寄存器。但该寄存器是可读的，以验证处理器相应的操作是否正确。

为了简化测试，Random 寄存器在系统重起时置为上界。另外，当 Wired 寄存器被写时，该寄存器也要置为上界。

图 5-2 表示 Random 寄存器的格式，而表 5-3 描述 Random 寄存器各域的含义。



图 5-2 Random 寄存器

表 5-3 Random 寄存器各域

域	描述
Random	随机 TLB 索引值
0	保留。必须按 0 写入，读时返回 0。

5.3 EntryLo0 (2)以及 EntryLo1 (3)寄存器

EntryLo 寄存器包括两个相同格式的寄存器：

- EntryLo0 用于偶虚页
- EntryLo1 用于奇虚页

EntryLo0 和 EntryLo1 寄存器都是可读/写寄存器。当执行 TLB 读和写操作时，它们分别包括 TLB 项中奇偶页的物理页号（PFN）。图 5-3 表示这些寄存器的格式。

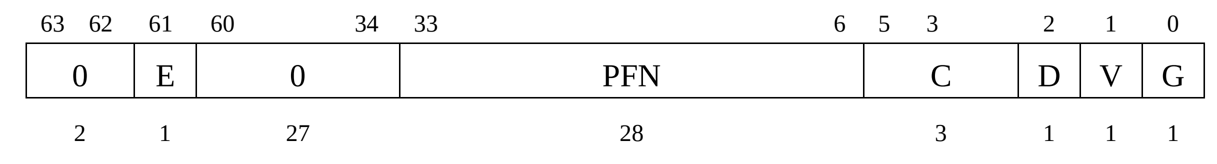


图 5-3 EntryLo0 和 EntryLo1 寄存器

EntryLo0 和 EntryLo1 寄存器的 PFN 域是 40 位物理地址中的高 28 位（39：12）。

表 5-4 EntryLo 寄存器域

域	描述
E	不可执行位。1 表示不可执行，0 表示可执行。
PFN	页号，是物理地址的高位。
C	TLB 页的 Cache 一致性属性。
D	脏位。如果该位被设置，页面则标记为脏，也就是可写的。实际上这一位在软件中作为防止数据被更改的写保护使用。
V	有效位。当该位被设置时，说明 TLB 表项是有效的，否则将产生一个 TLBL 或 TLBS 例外。

G	全局位。当 EntryLo0 和 EntryLo1 中的 G 位都被设置为 1 时，处理器将在 TLB 查找时忽略 ASID。
0	保留。必须按 0 写入，读时返回 0。

在每个 TLB 表项中只有一个全局位，在 TLB 写操作中根据 EntryLo0[0]和 EntryLo1[0]的值写入。

5.4 Context (4)

Context 寄存器是一个读/写寄存器，它包含指向页表中某一项的指针。该页表是一个操作系统数据结构，存储虚拟地址到物理地址的转换。

当 TLB 缺失时，CPU 将根据缺失转换从页表中加载 TLB。一般情况下，操作系统使用 Context 寄存器寻址页表中当前页的映射。Context 寄存器复制 BadVAddr 寄存器中的部分信息，但是该信息被安排成一种利于软件 TLB 例外处理程序处理的形式。

图 5-4 显示了 Context 寄存器的格式；表 5-5 描述了上下文寄存器字段。

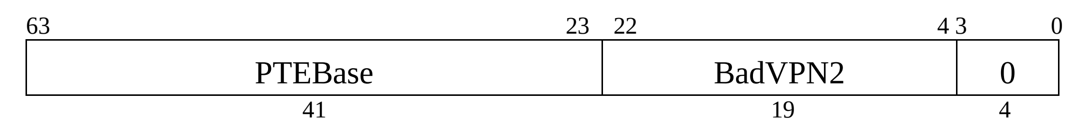


图 5-4 Context 寄存器

表 5-5 Context 寄存器域

域	描述
BadVPN2	当缺失时这一字段被硬件写。它包含最近不能进行有效转换的虚地址的虚页号(VPN)。
PTEBase	这一字段是操作系统使用的读/写字段。该字段写入的值允许操作系统将 Context 寄存器作为一个指向内存中当前页表的指针。
0	保留。必须按 0 写入，读时返回 0。

19 位的 BadVPN2 字段包含导致 TLB 缺失的虚地址的 31:13 位；第 12 位被排除是因为一个单一的 TLB 项映射到一个奇偶页对。对于一个 4K 字节的页尺寸，这一格式可以直接寻址 PTE 表项为 8 字节长且按对组织的页表。对于其它尺寸的页和 PTE，移动和屏蔽这个值可以产生合适的地址。

5.5 PageMask 寄存器(5)

PageMask 寄存器是个可读写的寄存器，在读写 TLB 的过程使用；它包含一个比较掩码，可为每个 TLB 表项设置不同的页大小，如表 5-6。该寄存器的格式如图 5-5。

TLB 读写操作使用该寄存器作为一个源或目的；当进行虚实地址转换时，TLB 中对应于 PageMask 寄存器的相应位指示虚地址位 24:13 中哪些位用于比较。当 MASK 域的值不是表 5-6 中的值时，TLB 的操作为未定义。0 域为保留，必须按 0 写入，读时返回 0。

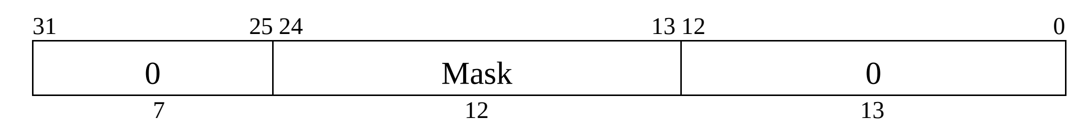


图 5-5 PageMask 寄存器

表 5-6 不同页大小的掩码（Mask）值

页大小	位											
	24	23	22	21	20	19	18	17	16	15	14	13
4Kbytes	0	0	0	0	0	0	0	0	0	0	0	0
16 Kbytes	0	0	0	0	0	0	0	0	0	0	1	1
64 Kbytes	0	0	0	0	0	0	0	0	1	1	1	1
256 Kbytes	0	0	0	0	0	0	1	1	1	1	1	1
1 Mbytes	0	0	0	0	1	1	1	1	1	1	1	1
4 Mbytes	0	0	1	1	1	1	1	1	1	1	1	1
16M bytes	1	1	1	1	1	1	1	1	1	1	1	1

5.6 Wired 寄存器(6)

Wired 寄存器是一个可读/写的寄存器，该寄存器的值指定了 TLB 中固定表项与随机表项之间的界限，如图 5-6 所示。Wired 表项是固定的、不可替换的表项，这些表项的内容不会被 TLB 写操作修改。而随机表项的内容可以被修改。

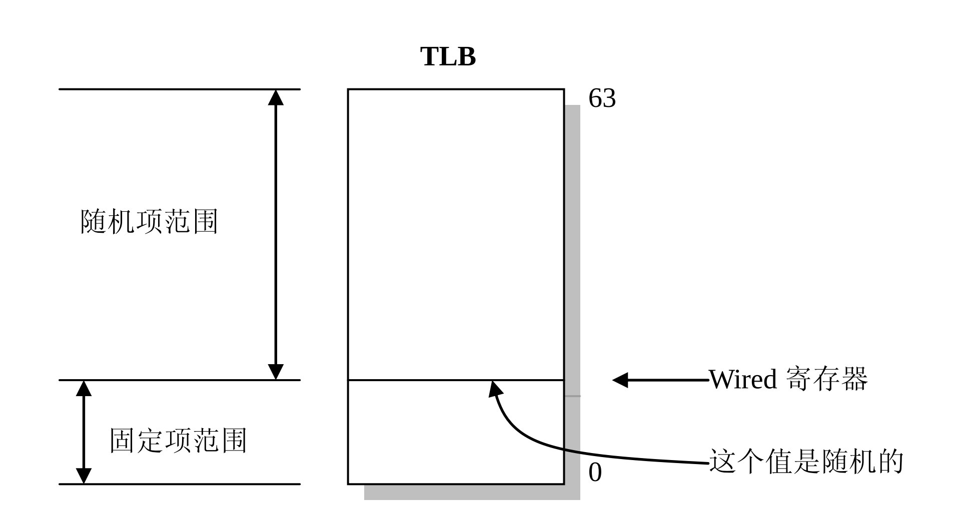


图 5-6 Wired 寄存器界限

Wired 寄存器在系统复位时置 0。写该寄存器的同时，Random 寄存器值要置为上限值（参阅前面的 Random 寄存器）。

图 5-7 表示 Wired 寄存器的格式；表 5-7 描述了寄存器的域。

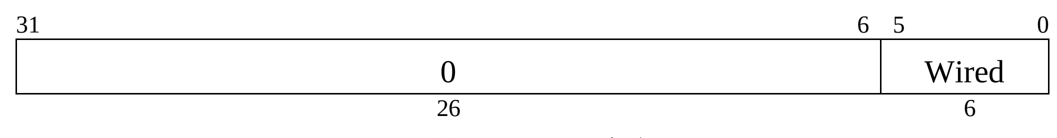


图 5-7 Wired 寄存器

表 5-7 Wired 寄存器域

域	描述
Wired	TLB 固定表项边界
0	保留。必须按 0 写入，读时返回 0。

5.7 BadVAddr 寄存器(8)

错误虚地址寄存器（BadVAddr）是一个只读寄存器，它记录了最近一次导致 TLB 或寻址错误例外的虚拟地址。除非发生软件复位，NMI 或 Cache 错误例外，BadVAddr 寄存器将一直保持不变。否则这个寄存器就是未定义的。

图 5-8 显示了错误虚地址寄存器的格式。

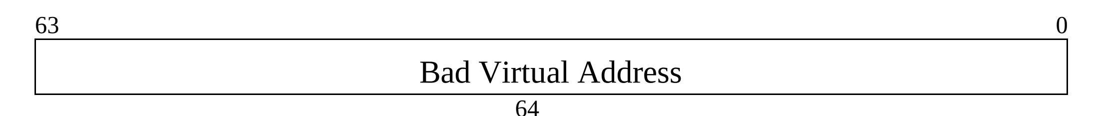


图 5-8 BadVAddr 寄存器

5.8 Count 寄存器(9)以及 Compare 寄存器(11)

Count 寄存器和 Compare 寄存器都是 32 位读写寄存器，他们的模式如图 5-9 所示。

Count 寄存器作为一个实时的定时器工作，每两个时钟周期增 1。

Compare 寄存器用来在特定的时刻生成一个中断，该寄存器被写入一个值，并且不断地与 Count 寄存器中的值比较。一旦这两个值相等，Cause 寄存器里的中断位 IP[7]被设置。当 Compare 寄存器被再次写时这个中断位才被重置。

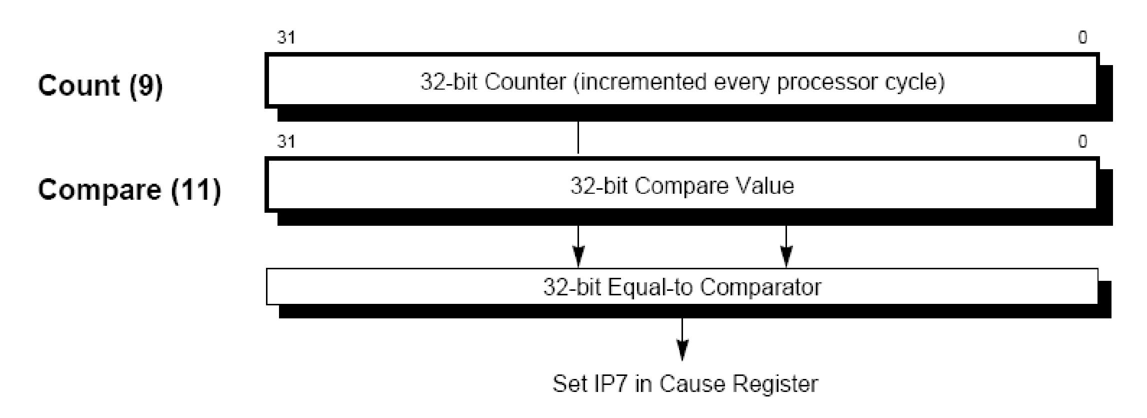


图 5-9 Count 寄存器和 Compare 寄存器

5.9 EntryHi 寄存器(10)

EntryHi 寄存器用于 TLB 读写时存放 TLB 表项的高位。

EntryHi 寄存器可以被 TLB Probe, TLB Write Random, TLB Write Indexed, 和 TLB Read Indexed 指令访问。

图 5-10 表示 EntryHi 寄存器的格式。表 5-8 表示 EntryHi 寄存器的域。

63 62	61 13	12 8	7 0
R	VPN2	0	ASID
2	49	5	8

图 5-10 EntryHi 寄存器

表 5-8 EntryHi 寄存器域

域	描述
VPN2	虚页号除 2（映射到双页）；虚拟地址的高位。
ASID	地址空间标识域。一个 8 位的域；用于让多个进程共享 TLB；对于相同的虚页号，每个进程都与其他进程有不同的映射。
R	区域位。（00->用户，01->超级用户，11->核心）用于匹配 vAddr63...62
0	保留。必须按 0 写入，读时返回 0。

VPN2 域包含 64 位虚拟地址的 61:13 位。

当一个 TLB Refill，TLB Invalid，或 TLB Modified 例外发生时，没有匹配 TLB 表项的虚拟地址中虚拟页号（VPN2）和 ASID 将被加载到 EntryHi 寄存器。

5.10 Status 寄存器(12)

Status 寄存器(SR) 是一个读写寄存器，它包括操作模式，中断允许和处理器状态诊断。下面列表描述了一些更重要的 Status 寄存器字段；图 5-11 显示了整个寄存器的格式，包括域的描述。其中重要的域有：

- 8 位的中断屏蔽(IM)域控制 8 个中断条件的使能。中断在被触发之前必须被使能，在 Status 寄存器的中断屏蔽域和 Cause 寄存器的中断待定域相应的位都应该被置位。更多的信息，请参考 Cause 寄存器的中断待定（IP）域。
- 4 位的协处理器可用性（CU）域控制 4 个可能的协处理器的可用性。不管 CU0 位如何设置，在内核模式下 CP0 总是可用的。

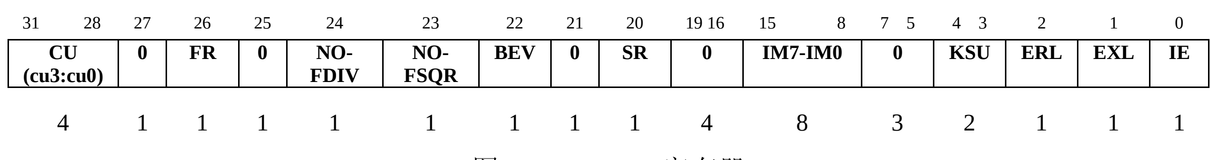


图 5-11 Status 寄存器

Status 寄存器格式

图 5-11 显示了 Status 寄存器的格式，表 5-9 描述了 Status 寄存器的域。

表 5-9 Status 寄存器域

域	描述
CU	控制 4 个协处理器单元的可用性。不管 CU0 位如何设置，在内核模式下 CP0 总是可用的。 1- 可用 0- 不可用 CU 域的初值是 0011
0	保留。必须按 0 写入，读时返回 0。
FR	使能附加的浮点寄存器 s 0 - 16 个寄存器 1 - 32 个寄存器
NOFDIV	禁止除法功能部件 1 - 禁止 0 – 允许
NOFSQR	禁止开根功能部件 1 - 禁止 0 – 允许
BEV	控制例外向量的入口地址 0 – 正常 1 – 启动
SR	1 表示有软复位例外发生
IM	中断屏蔽：控制每一个外部、内部和软件中断的使能。如果中断被使能，将允许它触发，同时 Cause 寄存器的中断 Pending 字段相应的位被置位。 0—禁止 1—允许
KSU	模式位 11 → 未定义 10 → 普通用户 01 → 超级用户 00 → 核心
ERL	错误级。当发生复位，软件复位，NMI 或 Cache 错误时处理器将重置此位。 0 → 正常 1 → 错误
EXL	例外级。当一个不是由复位，软件复位或 Cache 错误引发的例外产生时，处理器将设置该位。
IE	中断使能。

	0 → 禁用所有中断 1 → 使能所有中断

Status 寄存器模式和访问状态

下面描述 Status 寄存器中用于设置模式和访问状态的域：

- **中断使能：**当符合以下条件时，中断被使能：
  - IE = 1 且
  - EXL = 0 且
  - ERL = 0。

如果遇到这些条件，IM 位的设置允许中断。

- **操作模式：**当处理器处于普通用户、内核和超级用户模式时需要设置下述位域。
  - 当 KSU = 10_2, EXL = 0 和 ERL = 0 时处理器处于普通用户态模式下。
  - 当 KSU = 01_2, EXL = 0 和 ERL = 0 时处理器处于超级用户态模式下。
  - 当 KSU = 00_2, or EXL = 1 或者 ERL = 1 时处理器处于内核态模式下。
- **内核地址空间访问：**当处理器处在内核模式时，可以访问内核地址空间。
- **超级用户地址空间访问：**当处理器处在内核模式或超级用户模式时，可以访问超级用户地址空间。
- **用户地址空间访问：**处理器在这三种操作模式下都可以访问用户地址空间。

Status 寄存器复位

复位时，Status 寄存器的值是 0x30400004。

5.11 Cause 寄存器(13)

32 位的可读写 Cause 寄存器描述了最近一个例外发生的原因。

图 5-12 显示了这一寄存器的域，表 5-10 描述了 Cause 寄存器的域。一个 5 位例外码（ExcCode）指出了原因之一，如表 5-11 所示。

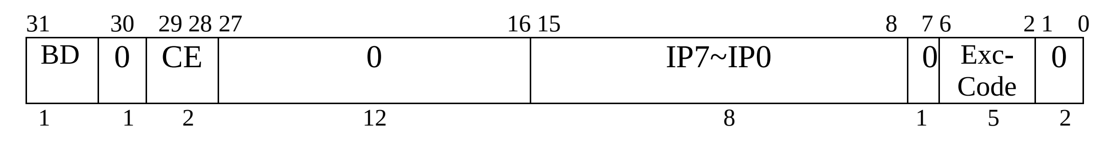


图 5-12 Cause 寄存器

表 5-10 Cause 寄存器域

域	描述
BD	指出最后采用的例外是否在分支延时槽中。 1—延时槽 0—正常

CE	当发生协处理器不可用例外时协处理器的单元编号。
IP	指出等待的中断。该位将保持不变直到中断撤除。IP0~IP1 是软中断位，可由软件设置与清除。 1－中断等待 0－没有中断
ExcCode	例外码域（见表5-11)
0	保留。必须按 0 写入，读时返回 0。

表 5-11 Cause 寄存器的 ExcCode 域

例外代码	Mnemonic	描述
0	INT	中断
1	MOD	TLB 修改例外
2	TLBL	TLB 例外（读或者取指令）
3	TLBS	TLB 例外（存储）
4	ADEL	地址错误例外（读或者取指令）
5	ADES	地址错误例外（存储）
6	IBE	总线错误例外（取指令）
7	DBE	总线错误例外（数据引用：读或存储）
8	SYS	系统调用例外
9	BP	断点例外
10	RI	保留指令例外
11	CPU	协处理器不可用例外
12	OV	算术溢出例外
13	TR	陷阱例外
14	-	保留
15	FPE	浮点例外
16－22	-	保留
23	WATCH	WATCH 例外
24－30	-	保留
31	-	保留

5.12 Exception Program Counter 寄存器(14)

例外程序计数器（Exception Program Counter，简称 EPC）是一个读/写寄存器，它包括例外处理结束后的继续处理地址。

对于同步例外，EPC 寄存器的内容是下面之一：

- 指令虚地址，这是导致例外的直接原因，或者

● 之前的分支或者跳转指令（当指令在分支延时槽中，指令延时位在 Cause 寄存器中被置位）的虚地址。

当 Status 寄存器中的 EXL 位被置 1 时，处理器不写 EPC 寄存器。

图 5-13 显示了 EPC 寄存器的格式。

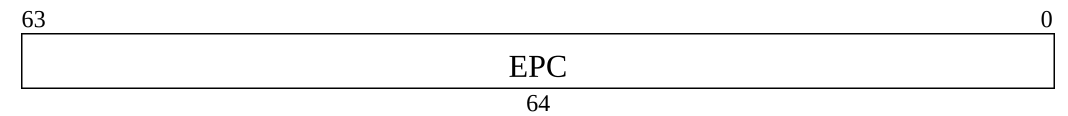


图 5-13 EPC 寄存器

5.13 Processor Revision Identifier (PRID)寄存器(15)

PRID 寄存器是个 32 的只读寄存器，该寄存器包含了标定处理器和 CP0 版本的实现版本和修订版本的信息。图 5-14 表示了该寄存器的格式；表 5-12 描述了该寄存器的域。

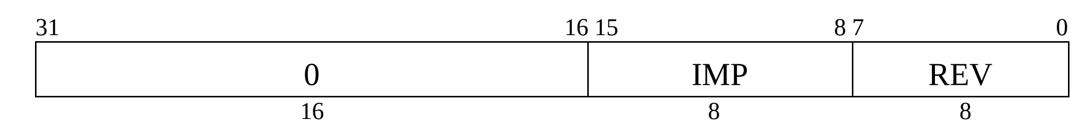


图 5-14 Processor Revision Identifier 寄存器

表 5-12 PRId 寄存器域

域	描述
IMP	实现版本号
REV	修订版本号
0	保留。必须按 0 写入，读时返回 0。

PRID 寄存器的低位（7：0 位）可用作修订版本的号码，而高位（15：8）位可用作实现版本的号码。龙芯 2F 实现版本号为 0x63，修订版本号为 0x02。

版本号码的表示格式为 Y.X，其中 Y（7：4 位）为主要版本号，而 X（3：0 位）为小版本号。

版本号码可以区分一些处理器的版本，但不能保证处理器的任何改动要体现在 PRID 寄存器中，换句话说，不能保证版本号的改动必须体现处理器的修改。因为这个原因，寄存器的值没有给出，而软件也不能依赖 PRID 寄存器中的版本号来标识处理器。

5.14 Config 寄存器(16)

Config 寄存器规定了龙芯 2F 处理器中各种配置选择项；表 5-13 列出了这些选项。

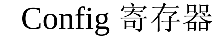
由 Config 寄存器的位 31:3 所定义的一些配置选项，在复位时由硬件设置，而且作为只读状态位包括在 Config 寄存器中，用于软件的访问。其他配置选项（Config 寄存器的位 2:0）是可读/写的并且由软件所控制。在复位时这些域是没有定义的。

Config 寄存器的配置是受限的。Config 寄存器在 Cache 被使用之前应该由软件来初



始化，并且，在做了任何改变后 Cache 应该重新初始化。

图 5-15 表示了 Config 寄存器的格式；表 5-13 描述了 Config 寄存器的域。Config 寄存器的初值为 0x00030932。



Config 寄存器

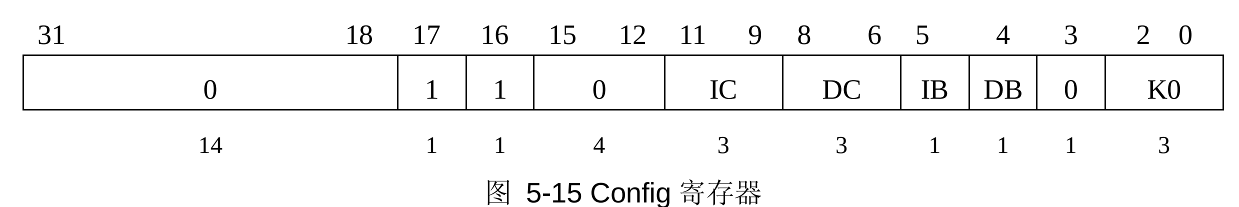


图 5-15 Config 寄存器

表 5-13 Config 寄存器域

域	描述
0	保留。必须按 0 写入，读时返回 0。
1	保留。必须按 1 写入，读时返回 1。
IC	一级指令 Cache 大小 (I-cache size = 2^{12+IC} Bytes)。龙芯 2F 为 64KByte。
DC	一级数据 Cache 大小 (D-cache size = 2^{12+DC} Bytes)。龙芯 2F 为 64KByte。
IB	一级 I-cache 行大小 0 → 16 字节 1 → 32 字节 龙芯 2F 该位设为 1
DB	一级 D-cache 行大小 0 → 16 字节 1 → 32 字节 龙芯 2F 该位设为 1
K0	Kseg0 的 Cache 一致性算法。 7 - Uncached Accelerated 3 - Cachable 2 - Uncached

5.15 Load Linked Address (LLAddr)寄存器(17)

可读/写寄存器，在龙芯 2F 中该寄存器无定义。

5.16 Watch 寄存器(18)

Watch 寄存器是 64 位的寄存器，包含位于虚地址空间一个双字的虚地址。如果被使能，任何读或写这个位置都将引发一个 Watch 例外。这个特性是为了调试使用的。

图 5-16 描述 Watch 寄存器的格式，表 5-14 描述了 Watch 寄存器的域。

Watch 寄存器

63 3 2 1 0



图 5-16 Watch 寄存器

表 5-14 Watch 寄存器域

域	描述
VADDR	虚地址的 63：3 位
R	如果设成 1，在 Load 时发生例外。
W	如果设成 1，在 Store 时发生例外。
0	保留。必须按 0 写入，读时返回 0。

5.17 XContext 寄存器(20)

可读写的 XContext 寄存器包含了一个指向操作系统页表中一个表项的指针。当发生一个 TLB 缺失时，操作系统根据缺失的转换从页表加载 TLB。

XContext 寄存器用于 XTLB 重填处理，处理 64 位地址空间的 TLB 表项加载，并仅供操作系统使用。操作系统根据需要设置寄存器中的 PTEBase 域。

图 5-17 显示了 XContext 寄存器的格式；表 5-15 描述了 XContext 寄存器的域。

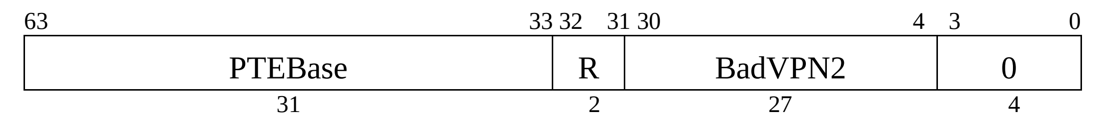


图 5-17 XContext 寄存器

27 位的 BadVPN2 域包含着引起 TLB 缺失的虚拟地址中 39：13 位，因为单一 TLB 表项映射到一个奇偶页对，所以第 12 位并没有被包括在内。当页面大小是 4K 字节时，这一格式可以直接寻址 PTE 表项为 8 字节长且按对组织的页表。对于其他的页面和 PTE 大小，经过移位和掩码可以得到正确的地址。

表 5-15 XContext 寄存器域

域	描述
BadVPN2	当发生缺失时硬件将写这个域，它包含了最近无效虚地址的虚页号除 2。
R	该域包含虚地址的 63:62 位。 00 = 普通用户 01 = 超级用户 11 = 内核
0	保留。必须按 0 写入，读时返回 0。
PTEBase	可读/写域，该值允许操作系统使用 XContext 寄存器作为一个指向内存中当前页表的指针。



5.18 Diagnostic 寄存器(22)

龙芯 2 号处理器新加的 64 位寄存器，主要用于控制处理器的一些内部队列和特殊操作。

Diagnostic 寄存器

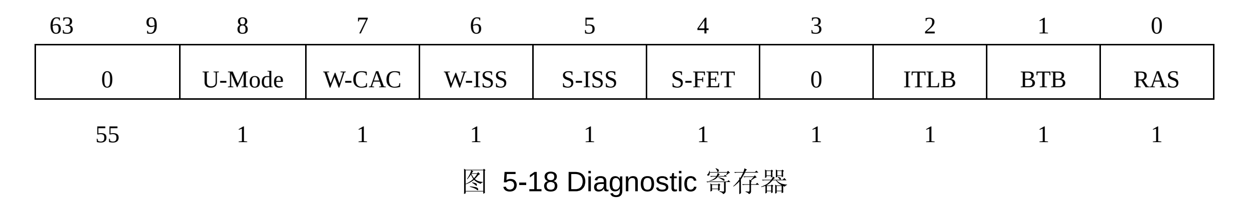


图 5-18 Diagnostic 寄存器

表 5-16 Diagnostic 寄存器域

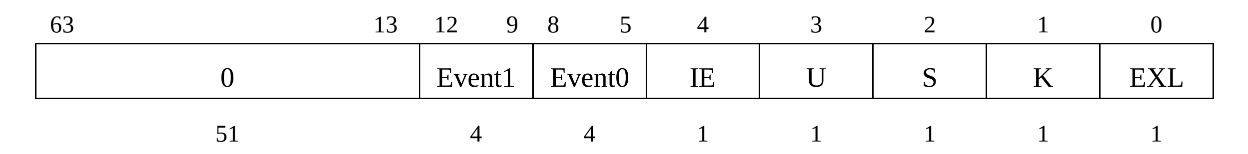
域	描述
0	保留。必须按 0 写入，读时返回 0。
U-Mode	用户态可执行 Cache Invalid 操作和 Writeback 操作
W-CAC	取消 Wait-Cache 操作的限制
W-ISS	取消 Wait-Issue 操作的限制
S-ISS	取消 Store-Issue 操作的限制
S-FET	取消 Store-Fetch 操作的限制
ITLB	写入 1 时清空 ITLB
BTB	写入 1 时清空 BTB
RAS	写入 1 时禁止使用 RAS。

5.19 Performance Counter 寄存器(24，25)

龙芯 2F 处理器定义了两个性能计数器（Performance Counter），他们分别映射到 CP0 寄存器的 24 号和 25 号。关联控制域位于 CP0 中 24 号寄存器。

每个计数器都是 32 位的读/写寄存器，并且在每次关联控制域中可数事件发生时自增。每个计数器都可以独立对一种事件计数。

性能计数寄存器 24 号



性能计数寄存器 25 号

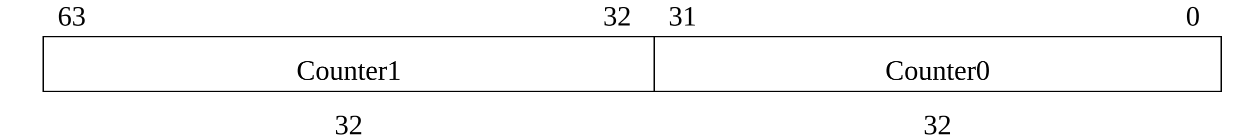


图 5-19 性能计数器寄存器

当计数器的首位（31 位）变成 1（计数器溢出）时，计数器将触发一个中断 IP[6]，关联控制域使能中断。在计数器溢出后无论中断是否被告知，计数都将继续。表 5-17 描述 24 号寄存器控制域的格式。表 5-18 描述计数使能位的定义。表 5-19 和表 5-20 描述计数器 0 和计数器 1 各自的事件。

表 5-17 控制域格式

[12:9]	[8:5]	[4]	[3:0]
Event 1 Select	Event 0 Select	IP[6] Interrupt Enable	计 数 使 能 位 (K/S/U/EXL)

表 5-18 计数使能位定义

计数使能位	Count Qualifier(CP0 Status 寄存器域)
K	KSU = 0 (内核模式), EXL = 0, ERL = 0
S	KSU = 1 (超级用户模式), EXL = 0, ERL = 0
U	KSU = 2 (普通用户模式), EXL = 0, ERL = 0
EXL	EXL = 1, ERL = 0

表 5-19 计数器 0 事件

事件	内部信号	描述
0000	Cycles	周期
0001	Brbus.valid	分支指令
0010	Jrcount	JR 指令
0011	Jr31count	JR 指令并且域 rs=31
0100	Imemread.valid& Imemread_allow	一级 I-cache 缺失
0101	Rissuebus0.valid	Alu1 操作已发射
0110	Rissuebus2.valid	Mem 操作已发射
0111	Rissuebus3.valid	Falu1 操作已发射
1000	Brbus_bht	BHT 猜测指令
1001	Mreadreq.valid& Mreadreq_allow	从主存中读
1010	Fxqfull	固定发射队列满的次数
1011	Roqfull	重排队列满的次数
1100	Cp0qfull	CP0 队列满的次数
1101	Exbus.ex & Excode=34,35	Tlb 重填例外

事件	内部信号	描述
1110	Exbus.ex & Excode=0	例外
1111	Exbus.ex & Excode=63	内部例外

表 5-20 计数器 1 事件

事件	内部信号	描述
0000	Cmtbus?.valid	提交操作
0001	Brbus.brerr	分支预测失败
0010	Jrmiss	JR 预测失败
0011	Jr31miss	JR 且 rs=31 预测失败
0100	Dmemread.valid& Dmemread_allow	一级 D-cache 缺失
0101	Rissuebus1.valid	Alu2 操作已发射
0110	Rissuebus4.valid	Falu2 操作已发射
0111	Duncache_valid& Duncache_allow	Uncached 访问
1000	Brbus_bhtmiss	BHT 猜测错误
1001	Mwritereq.valid& Mwritereq_allow	写到主存
1010	Ftqfull	浮点指针队列满的次数
1011	Brqfull	分支队列满的次数
1100	Exbus.ex & Op==OP_TLBPI	Itlb 缺失
1101	Exbus.ex	例外总数
1110	Mispec	Load 投机缺失
1111	CP0fwd_valid	CP0 队列向前加载

5.20 TagLo(28)和 TagHi (29)寄存器

TagLo 和 TagHi 寄存器是 32 位读/写寄存器，用于保存一级/二级 Cache 的标签和状态，使用 CACHE 和 MTC0 指令往 Tag 寄存器写。

图 5-20 显示了这些寄存器用于一级 Cache（P-Cache）操作的格式。表 5-21 列出了 TagLo 和 TagHi 寄存器中域的定义。

TagLo 寄存器

31 … 8	7 … 6	5 … 4	3 … 0
PTAG[23：0]	CS	SCSETI	0
24	2	2	4



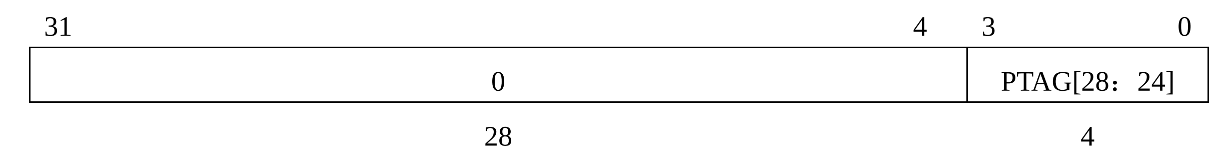


图 5-20 TagLo 和 TagHi 寄存器(P-Cache)

表 5-21 Cache Tag 寄存器域

域	描述
PTAG	指定物理地址的 39:12 位。
CS	指定 Cache 的状态。
SCSETI	对应 Cache 行在二级 Cache 的组号（二级 Cache 该域为 0）
0	保留。必须按 0 写入，读时返回 0。

5.21 ErrorEPC 寄存器(30)

除了用于 ECC 和奇偶错误例外外，ErrorEPC 寄存器与 EPC 寄存器类似。它用于在复位、软件复位、和不可屏蔽中断（NMI）例外时存储程序计数器。

ErrorEPC 是一个读写寄存器，它包括处理一个错误后指令重新开始执行的虚拟地址。图 5-21 显示了 ErrorEPC 寄存器的格式。

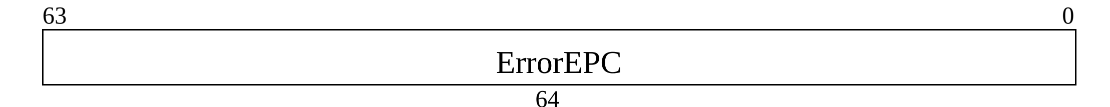


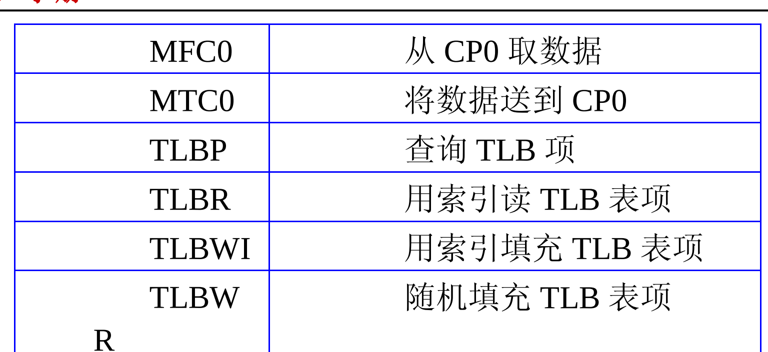
图 5-21 ErrorEPC 寄存器

5.22 CP0 指令

表 5-22 列出了 Godson-2 处理器定义的 CP0 指令。

表 5-22 CP0 指令

指令	描述
CACHE	CACHE 操作
DMFC0	从 CP0 取双字
DMTC0	将双字送到 CP0
ERET	例外返回



MFC0	从 CP0 取数据
MTC0	将数据送到 CP0
TLBP	查询 TLB 项
TLBR	用索引读 TLB 表项
TLBWI	用索引填充 TLB 表项
TLBWR	随机填充 TLB 表项

相关

龙芯处理器能够处理硬件中的流水线相关，包括 CP0 相关和访存相关，因此 CP0 指令并不需要 NOP 指令来校正指令序列。

6 处理器例外

本章介绍龙芯 2F 处理器例外，内容包括：例外的产生及返回，例外向量位置和所支持的例外类型。其中对每一类支持的例外类型，介绍内容包括例外的原因，处理和服务。

6.1 例外的产生及返回

当处理器开始处理某个例外，状态寄存器的 EXL 位是被置为 1 的，这意味着系统运行在内核模式。在保存了适当的现场状态之后，例外处理程序通常将状态寄存器的 KSU 字段设定为内核模式，同时将 EXL 位置回为 0。当恢复现场状态并且重新执行时，处理程序则会把 KSU 字段恢复回上次的值，同时置 EXL 位为 1。

从例外返回也会将 EXL 位置为 0。

6.2 例外向量位置

冷重置、软重置和非屏蔽中断(NMI)例外的向量地址是专用的冷重置例外向量地址，这个地址既不通过 Cache 进行存取，也无需地址映射。而所有其它例外向量地址的形式都是基地址加上向量偏移地址。

启动时(Boot-Time)例外向量(状态寄存器中的 BEV 位=1)地址位于既不通过 Cache 进行存取，也无需地址映射的地址空间。在正常操作期间(BEV 位=0)，普通例外的向量地址位于需要通过 Cache 进行存取的地址空间。

表 6-1 列出了龙芯 2F 处理器中例外的向量地址。

表 6-1 例外向量地址

BEV 位	例外类型	例外向量地址
	Cold Reset/ NMI	0xFFFFFFFF BFC00000
BEV = 0	TLB Refill (EXL=0)	0xFFFFFFFF 80000000
	XTLB Refill (EXL=0)	0xFFFFFFFF 80000000
	Others	0xFFFFFFFF 80000180
BEV = 1	TLB Refill (EXL=0)	0xFFFFFFFF BFC00200
	XTLB Refill (EXL=0)	0xFFFFFFFF BFC00200
	Others	0xFFFFFFFF BFC00380

6.3 TLB 重填例外向量选择

在当前所有 MIP III 指令集结构的实现中，存在两种 TLB 重填例外的向量：

- 一种是面向 32 位地址空间的（TLB 重填）
- 一种是面向 64 位地址空间的（XTLB 重填）

龙芯 2F 是 64 位的地址模式，因此采用 XTLB 重填例外。但为了兼容 32 位模式，龙芯 2F 的 XTLB 重填例外的向量入口地址和 TLB 重填例外相同，如表 6-1 所示。

6.4 例外优先级

这一章的剩余部分将按照表 6-2 中给出的优先顺序依次介绍各个例外(对某些特定例外，如 TLB 例外和指令/数据例外，为了方便而放在一起介绍)。当一条指令同时产生一个以上例外时，只向处理器报告其中优先级最高的例外。有些例外并不是由当时正在执行的指令产生的，而有些例外可能被推迟处理。更多细节请查看本章对各个例外的单独介绍。

表 6-2 例外优先顺序

例外优先顺序
冷重置(最高优先级)
不可屏蔽中断 (NMI)
地址错误 — 取指
TLB 重填 — 取指
TLB 无效 — 取指
总线错误 —取指
整型溢出，陷阱，系统调用，端点，保留指令，协处理器不可用，浮点例外
地址错误 — 数据存取
TLB 重填 — 数据存取
TLB 无效 — 数据存取
TLB 修改 — 写数据
Watch
总线错误 —数据存取
中断(最低优先级)

一般来说，下面各节中介绍的例外先由由硬件来处理，然后由软件来服务。

6.5 冷重置例外

原因

当系统第一次上电或者冷重置时，产生冷重置例外。该例外不可屏蔽。

处理

CPU 为这个例外提供了一个特殊的中断向量：

- 32 位模式下位于 0xBFC0 0000
- 64 位模式下位于 0xFFFF FFFF BFC0 0000

冷重置向量地址属于无需地址映射和不通过 Cache 存取数据的 CPU 地址空间，因此处理这个例外不必初始化 TLB 或 Cache。这也意味着即使 Cache 和 TLB 处于不确定状态，处理器也可以取出并执行指令。

当例外发生时，CPU 中所有寄存器内容是不确定的，但下列寄存器域除外：

● 状态（Status）寄存器的初值为 0x30400004，SR 位被清为 0，ERL 位和 BEV 位被置为 1。

● 配置（Config）寄存器的初值为 0x00030932。

● 随机（Random）寄存器初始化为它的最大值。

● Wired 寄存器初始化为 0。

● ErroEPC 寄存器初始化为 PC 的值。

● Performance Count 寄存器的 Event 位初始化为 0。

● 所有等待处理的 Watch 例外和外部中断都被清除。

服务

冷重置例外服务包括：

● 初始化所有的处理器寄存器，协处理器，Cache 和存储系统。

● 执行诊断测试。

● 引导自举操作系统。

6.6 NMI 例外

原因

NMIn 为低产生 NMI 例外。该例外不可屏蔽。

处理

当发生 NMI 例外时，状态寄存器的 SR 位被置为 1，用以区分冷重置。

NMI 例外只能在指令的边界被提取。它并不抛弃任何机器的状态，而是保留处理器的状态用于诊断。Cause 寄存器内容保持不变，而系统则跳到 NMI 例外处理程序开始处。

NMI 例外保留除下列寄存器外的所有寄存器值：

● 包含 PC 值的 ErrorEPC 寄存器。

● 置为 1 的状态寄存器 ERL 位。

● 软重置或 NMI 置为 1，冷重置置为 0 的状态寄存器 SR 位。

● 置为 1 的状态寄存器 BEV 位。

● PC 寄存器重置为 0xFFFF FFFF BFC0 0000

服务

NMI 例外可以用于除“重置处理器，同时保持 Cache 和内存内容”之外的情形。例如，当检测到电源故障时，系统可以通过 NMI 例外立即、可控地关闭系统。

由于 NMI 例外在另外一个错误例外中发生，因此从例外返回后，通常不太可能继续执行程序。

6.7 地址错误例外

原因

当执行以下情况时，会发生地址错误例外：

- 引用非法地址空间。
- 在用户模式下引用超级用户地址空间。
- 在用户或超级用户模式下引用内核地址空间。
- 取(Load)或存(Store)一个双字，但双字不对齐于双字边界。
- 取(Load，Fetch)或存(Store)一个字，但字不对齐于字的边界。
- 取或存一个半字，但半字不对齐于半字的边界。

该例外不可屏蔽。

处理

共用例外向量用于地址错误例外。Cause 寄存器的 ExcCode 字段值被设为 ADEL 或 ADES 编码值，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

例外发生时，BadVAddr 寄存器保存了没有正确对齐的虚地址，或者受保护的地址空间的虚地址。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

此时，正在运行的导致例外发生的进程会收到 UNIX SIGSEGV(段违例)信号，这个错误对该进程来说通常是致命的。

6.8 TLB 例外

可能发生三种 TLB 例外：

- 当 TLB 中没有项与欲引用的映射地址空间的地址匹配时，会导致 TLB 重填例外。
- 当虚地址引用与 TLB 中某一项匹配，但该项被标示为无效时，TLB 无效例外发生。
- 当写内存操作的虚地址引用与 TLB 中某项匹配，但该项并没有被标示为“脏”时（表示该项不可写），TLB 修改例外发生。

下面三节将介绍这些 TLB 例外。

注：TLB 重填向量选择已经在本章前面作了介绍，具体章节见 6.9“TLB 重填例外”。

6.9 TLB 重填例外

原因

当 TLB 中没有项匹配映射地址空间的引用地址时，TLB 重填例外发生，该例外是不可屏蔽的。

处理

对于这个例外来说，MIPS 体系结构存在两个特殊的例外向量：一个用于 32 位地址空间，另一个用于 64 位地址空间。龙芯 2F 为了兼容 32 位，这两个例外的入口一致，相当于两个例外同时存在，可根据需要处理其中一个。此时，处理器当前的操作模式并不重要，除非它参与了确定某一地址属于何种地址空间。如果地址属于 XUSEG、XSUSEG 或者 XKUSEG，则该地址属于用户地址空间。如果地址属于 CSSEG、CKSSEG、XSSEG 或者 XKSSEG，那么该地址就属于超级用户地址空间。而内核地址空间归属的确定则是看地址是否属于 CKSEG3 或者 XKSEG。CKSEG0、CKSEG1 以及内核物理地址空间（XKPHY）属于非地址映射的内核空间。

当状态寄存器中的 EXL 位被设为 0 时，所有的地址引用使用这些例外向量。这个例外设置 Cause 寄存器中 ExcCode 字段的值为 TLBL 或 TLBS 编码。这个编码与 EPC 寄存器以及 Cause 寄存器的 BD 一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

发生这个例外时，BadVAddr、Context、XContext 和 EntryHi 寄存器保存了那条地址转换失败的虚地址。EntryHi 寄存器也保存了转换失败时的 ASID。Random 寄存器通常保存了用于放置被替换 TLB 项的合法位置。EntryLo 寄存器的内容是不确定的。如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了导致例外的指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

为了服务这个例外，Contex 或 XContext 寄存器的内容被作为虚地址以取得某些内存位置，这些位置包含了一对 TLB 项的物理页地址和访问控制位。这一对 TLB 项被放入了 EntryLo0/EntryLo1 寄存器；EntryHi 和 EntryLo 寄存器被写入 TLB。

用于获得物理地址和访问控制信息的虚地址有可能位于一个没有驻留在 TLB 中的页面上。如果出现这种情况，则在 TLB 重填处理程序允许另外一个 TLB 重填例外来解决。由于 Status 寄存器的 EXL 位被置为 1，第二个 TLB 重填例外被传递的是共用例外向量。

6.10 TLB 无效例外

原因

当一个虚地址引用匹配到一项被标记为无效的 TLB 项(TLB 有效位被清掉)时，TLB

无效例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外。Cause 寄存器的 ExcCode 字段值被设为 TLBL 或 TLBS，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

发生这个例外时，BadVAddr、Context、XContext 和 EntryHi 寄存器保存了那条地址转换失败的虚地址。EntryHi 寄存器也保存了转换失败时的 ASID。Random 寄存器通常保存了用于放置被替换 TLB 项的合法位置。EntryLo 寄存器的内容是不确定的。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

当发生下面情况之一时，TLB 项被标记为无效：

- 虚地址不存在
- 虚地址存在，但是不在主存中(缺页)
- 引用这个页而引发一个陷阱(例如,维护引用位)

在服务完 TLB 无效例外的起因之后，通过 TLBP 指令来定位 TLB 项(探测 TLB 来找匹配的项)，然后用标记位有效的一项来替换该 TLB 项。

6.11 TLB 修改例外

原因

当写内存操作的虚地址引用与 TLB 中某项匹配，但该项并没有被标示为“脏”，因此该项不可写时，TLB 修改例外发生。该例外不可屏蔽。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器中的 ExcCode 字段值被设置为 MOD。

发生这个例外时，BadVAddr、Context、XContext 和 EntryHi 寄存器保存了那条地址转换失败的虚地址。EntryHi 寄存器也保存了转换失败时的 ASID。EntryLo 寄存器的内容是不确定的。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

内核使用失败的虚地址或虚页号来识别相应的访问控制信息。被识别的页可能允许

或者不允许写访问；如果写访问不允许，那么写保护违例发生。

如果写访问是允许的，那么内核在其自己的数据结构内将页标记为可写。TLBP 指令把必须改变的 TLB 项的索引放入到 Index 寄存器中。包含物理页和访问控制位(D 位被设置)的一个字被取出放入 EntryLo 寄存器中，然后，EntryHi 和 EntryLo 寄存器被写入 TLB 中。

6.12 总线错误例外

原因

当处理器进行数据块的读取、更新或双字/单字/半字的读请求时收到外部的 ERR 完成应答信号，或者处理器双字/单字/半字的读请求时收到外部的 ACK 完成应答信号，并且与之相关联的双字/单字/半字数据应答包含了不可改正的错误，总线错误例外发生。该例外不可屏蔽。

处理

共用中断向量用于处理总线错误例外。Cause 寄存器的 ExcCode 字段值被设为 IBE 或 DBE，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

发生错误的物理地址可以通过 CP0 寄存器中的信息计算出来。

如果 Cause 寄存器中的 ExcCode 字段值被设置为 IBE 编码(表示是取指引用)，那么导致例外发生的指令虚地址保存在 EPC 寄存器中（如果 Cause 寄存器的 BD 位被置为 1，则该指令的虚地址为 EPC 寄存器内容加 4）。

如果 Cause 寄存器中的 ExcCode 字段值被设置为 DBE 编码(表示是读取或存储引用)，那么导致例外发生的指令虚地址保存在 EPC 寄存器中（如果 Cause 寄存器的 BD 位被置为 1，则该指令的虚地址为 EPC 寄存器内容加 4）。

于是，读取和存储引用的虚地址就可以通过解释这条指令来获得。而物理地址可以通过 TLBP 指令以及读取 EntryLo 寄存器内容计算物理页号来获得。导致例外发生的正在运行的进程会收到 UNIX SIGBUS(总线错误)信号，对该进程来说这通常是致命的。

6.13 整型溢出例外

原因

当一条 ADD、ADDI、SUB、DADD、DADDI 或 DSUB 指令执行，导致结果的补码

溢出时，整型溢出例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 OV 编码值。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

导致例外发生的正在执行的进程会收到一个 UNIX SIGFPE/FPE_INTOVE_TRAP(浮点例外/整型溢出)信号。对该进程来说，这个错误通常是致命的。

6.14 陷阱例外

原因

当 TGE、TGUE、TLT、TLTU、TEQ、TNE、TGEI、TGEUI、TLTI、TLTUI、TEQI、TNEI 指令执行，条件结果为真时，陷阱例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 TR 编码值。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

导致例外发生的正在执行的进程会收到一个 UNIX SIGFPE/FPE_INTOVE_TRAP（浮点例外/整型溢出）信号。对该进程来说，这个错误通常是致命的。

6.15 系统调用例外

原因

当执行 SYSCALL 指令的时候，系统调用例外发生。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 SYS 编码值。

如果 SYSCALL 指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令的地址。

如果 SYSCALL 指令在分延迟槽中，则状态寄存器中的 BD 位被置为 1，否则该位被

清 0。

服务

当这个例外发生时，控制权被转到适当的系统例程。进一步的系统调用区分可以分析 SYSCALL 指令的 Code 字段（位 25：6），以及载入 EPC 寄存器中所存地址的指令的内容。

为了恢复进程的执行，必须改变 EPC 寄存器的内容，这样 SYSCALL 指令才不会再次被执行；这可以通过在返回之前使 EPC 寄存器的值加 4 来完成。

如果 SYSCALL 指令处在分支延迟槽中，则需要更复杂的算法，这已经超出了本节所能描述的范围。

6.16 断点例外

原因

当执行一条 BREAK 指令时，发生断点例外。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 BP 编码值。

如果 BREAK 指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令的地址。

如果 BREAK 指令在分延迟槽中，则状态寄存器中的 BD 位被置为 1，否则该位清 0。

服务

当这个例外发生时，控制权被转到适当的系统例程。进一步的区分可以分析 BREAK 指令的 Code 字段（位 25：6），以及载入 EPC 寄存器中所存地址的指令的内容。如果这条指令在分支延迟槽中，那么 EPC 寄存器中的内容必须加上 4 以定位到该指令。

为了恢复进程的执行，必须改变 EPC 寄存器的内容，这样 BREAK 指令才不会再次被执行；这可以通过在返回之前使 EPC 寄存器的值加 4 来完成。

如果 BREAK 指令在分支延迟槽中，那么为了恢复进程的继续执行，需要解释这条分支指令。

6.17 保留指令例外

原因

当以下条件之一成立时，保留指令例外发生：

- 试图执行一条主操作码(位 31:26)没有定义的指令。
- 试图执行一条次操作码(位 5:0)没有定义的 SPECIAL 指令。
- 试图执行一条次操作码(位 20:16)没有定义的 REGIMM 指令。

- MIPS IV ISA 不可用时，试图执行一条 COP1X 指令。

这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 RI 编码值。

如果保留指令指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令地址。

服务

此时，MIPS IV ISA 中没有指令被解释执行。正在执行的导致例外发生的进程会收到 UNIX SIGILL/ILL_RESOP_FAULT(非法指令/保留的操作错误)信号。对该进程来说，这个错误通常是致命的。

6.18 协处理器不可用例外

原因

试图执行以下任意一条协处理器指令，将会导致协处理器不可用例外发生：

- 相应的协处理器单元（CP1 或 CP2）没有被标记为可用。
- CP0 单元没有被标记为可用，并且进程执行在用户或者超级用户的模式下的 CP0 指令。

这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 CPU 编码值。Cause 寄存器的 CE 域指示四个协处理器的哪个被引用。如果这条指令不是在分支延迟槽中，EPC 寄存器保存了不可使用协处理器指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址。

服务

有以下的几种情形：

如果进程被授权访问协处理器，协处理器被标记为可用，那么相应的用户状态被恢复以便协处理器执行。

如果进程被授权访问协处理器，但是协处理器不存在或者有故障，则需要解释/模拟这条协处理器指令。

如果在 Cause 寄存器中的 BD 位被设置了，分支指令必须被解释；然后协处理器指令被模拟。例外返回时跳过例外的协处理器指令继续执行。

如果进程没有被授权访问协处理器，这时执行的进程收到 UNIX SIGILL/ILL_PRIVIN_FAULT（非法指令/特权指令错误）信号。这个错误通常是致命的。

6.19 浮点例外

原因

浮点协处理器使用浮点例外。这个例外是不可屏蔽的。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 FPE 编码值。

浮点控制/状态寄存器的内容指示这个例外产生的原因。

服务

清除浮点/状态寄存器中的适当位可以清除这个例外。

6.20 Watch 例外

原因

当取或存指令引用了在系统控制协处理器(CP0)的寄存器 Watch 所设定的虚地址的时候，Watch 例外发生。Watch 寄存器的最低 2 位指定是读取或存储指令引发了这个例外。

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode 字段被置为 Watch 编码值。

服务

Watch 例外用于调试目的；通常例外处理程序把控制权转交给调试程序，并允许用户检查当前状态。

为了执行被调试的指令，必须暂时禁止 Watch 例外。为了继续调试，又必须重新激活 Watch 例外。可以通过解释执行被调试指令或者设断点的方式来完成这一过程。

6.21 中断例外

原因

当八个中断条件中的一个触发，中断例外发生。这些中断的重要性依赖于特定的系统实现。

通过清掉在状态寄存器中的 Interrupt-Mask (IM)域中的相应的位，八个中断中的任何一个都可以被屏蔽，并且，通过清掉状态寄存器的 IE 位，可以一次屏蔽所有的八个中断。

处理

共用例外向量用于处理该例外，并且 Cause 寄存器的 ExcCode 字段被置为 INT 编码值。

Cause 寄存器中的 IP 域指明了当前的中断请求。不止一个的中断位可能同时被设置

(如果中断触发并且在寄存器被读到之前被撤消，甚至没有位被设置)。

IP[7]中断为内部中断，在 Count 寄存器内容与 Compare 寄存器内容相等时产生中断；IP[6]中断位既被配置为第五个外部中断，也被配置为 CP0 性能计数器溢出中断。

软件需要对每一个可能的中断源进行查询来判断中断产生的原因（一个中断同时可能有多个源）。

服务

如果中断是由两个软件产生例外之一导致的，则设置 Cause 寄存器中的相应位，IP[1：0]，为 0 来清除中断条件。

软件中断是非精确的。一旦软件中断触发，在例外被处理之前，程序还可能继续执行几条指令。定时器中断的清除通过向 Compare 寄存器写入值来完成。性能计数器中断的清除则是向计数器的溢出位，即位 31，写入 0 来实现。

冷重置和软重置会清除所有未完成的外部中断请求，IP[2]至 IP[6]。

如果中断是硬件产生的，那么撤消引起触发的中断管脚的条件，就可清除中断条件。

7 浮点部件

本章描述了龙芯 2F 处理器浮点部件（Floting Point Unit，简称 FPU）的特性，包括编程模型、指令集和指令格式、指令流水线以及异常。龙芯 2F 浮点部件及其相关的系统软件完全符合 ANSI/IEEE 754－1985 二进制浮点运算标准。另外，龙芯 2F 浮点部件能够处理自定义 SIMD 多媒体定点指令集。

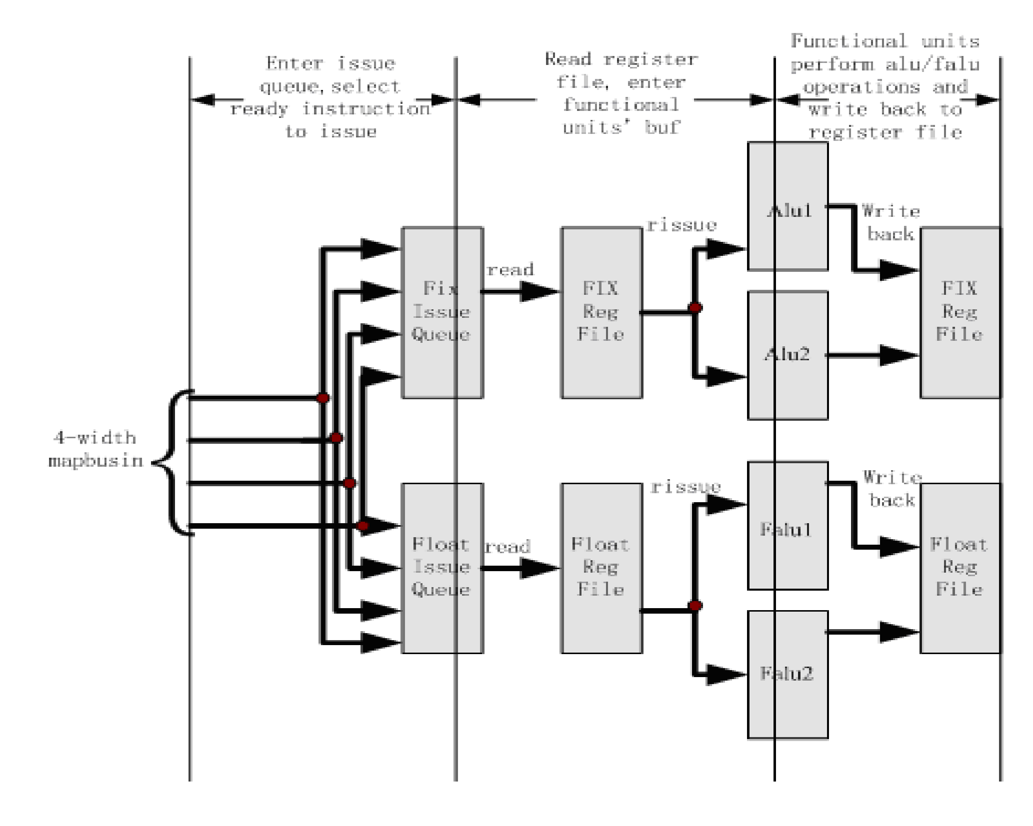
7.1 概述

FPU 作为 CPU 的协处理器，被称为 CP1（Coprocessor 1），通过扩展 CPU 的指令集来完成浮点算术运算功能。

FPU 由以下两个功能单元组成：

- FALU1 单元
- FALU2 单元

FALU1 单元执行浮点加减、浮点乘法、浮点乘加、取绝对值、取反、精度转换、定浮点格式转换、比较、转移等操作。FALU2 执行浮点加减、浮点乘法、浮点乘加、浮点除法、浮点开平方操作。另外，龙芯 2F 的 FPU 还可以执行并行单精度（Paired-Single，简称 PS）浮点指令和多媒体定点指令。图 7-1 对龙芯 2F 体系结构中功能单元的组织构成进行了图解说明。



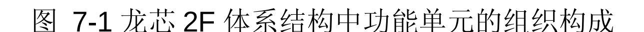


图 7-1 龙芯 2F 体系结构中功能单元的组织构成

浮点队列每个时钟周期可以分别发射 1 条指令到 FALU1 单元、1 条指令到 FALU2 单元。浮点寄存器文件为 FALU1 单元与 FALU2 单元各提供三个专用的读端口和一个专用的写端口。

7.2 FPU 编程模型

这部分描述 FPU 寄存器组和它们的数据组织结构。FPU 寄存器包括浮点通用寄存器（Floating-point General Registers，简称 FGRs）和两个控制寄存器：控制/状态（Control/Status）和实现/修订（Implementation/Revision）。

7.2.1 浮点寄存器

浮点单元是 MIPS IV 指令集体系结构中的协处理器 1 的硬件执行。MIPS IV 工业标准结构中定义了 32 个浮点逻辑通用寄存器（FGRs），每个 FGR 为 64 位宽，可以存放 32 位单精度数或 64 位双精度数。在浮点寄存器文件中，硬件上实际包含 64 个 64 位物理寄存器，而只使用其中 32 个作为逻辑寄存器。

浮点指令用一个 5 位的逻辑寄存器号进行选择 FGR。在浮点单元执行前，这些逻辑寄存器号被重命名单元（RegMap）映射为物理寄存器号。物理寄存器根据 6 位的地址数据进行选择。

状态寄存器中的 FR 位(26)决定了程序可见的浮点逻辑寄存器的个数，并影响单精度 Load/Store 指令的相应操作。.

● FR 为 0 时仅使用 16 个偶数号的 64 位逻辑寄存器或 32 个 32 位的逻辑寄存器，这与 MIPS I 和 MIPS II 工业标准结构兼容。

● FR 为 1 时所有的 32 个 64 位逻辑寄存器都是可用的。这是标准的 MIPS III 与 MIPS IV 操作。

7.2.2 浮点控制寄存器

MIPS IV 工业标准体系允许每个协处理器定义最多 32 个控制寄存器，但是龙芯 2F 的浮点部件只使用了两个：

● 控制寄存器 0，浮点实现和修订寄存器

● 控制寄存器 31，浮点状态寄存器(FSR)。

控制寄存器(FCRs) 只能被 CFC1/CTC1 操作访问。实现和修订寄存器（FCR0）保存 FPU 的修订信息，控制和状态寄存器（FCR31）控制和监视例外，保存比较运算的结果并且产生舍入模式。

实现和修订寄存器 (FCR0)

只读寄存器 FCR0 指定了 FPU 的实现和修订版本。它表明了协处理器 1 的修订情况

和性能级别，同时这些信息也可被诊断软件所利用。

表 7-1 描述了实现/修订寄存器（FCR0）的各个域。

表 7-1 FCR0 域

域	描述
IMP[15:8]	实现版本号（0x05）
REV[7:0]	以 y.x 形式表示的修订版本号（0x01）
0[31:16]	保留。必须被写入 0，当读取该域时返回 0

控制/状态寄存器（FCR31）

控制/状态寄存器（FCR31）包含了控制和状态信息，这些信息既能够在核心模式下被指令访问，也能在用户模式下被指令访问。FCR31 还控制着算术运算的舍入模式和用户模式下陷阱的使能，同时识别在最近执行的指令中可能发生的例外以及那些没有被捕获的可能发生的例外。

图 7-2 说明了控制/状态寄存器的格式，表 7-2 描述了控制/状态寄存器的各个域。



图 7-2 浮点控制/状态寄存器

表 7-2 控制/状态寄存器域

域	描述
CC7-CC1	控制位 7-1。当双单精度的高位部分比较运算结果为真，CC1 被置位。CC7-CC2 在龙芯 2F 中未定义。
FS	当设置此位后，亚正常结果被置为0，不再引起未实现操作例外。
CC0	条件位。参见控制/状态寄存器条件位的描述。
Causes	导致位。参见控制/状态寄存器导致位的描述。
Enables	使能位。参见控制/状态寄存器使能位的描述。
Flags	标志位。参见控制/状态寄存器标志位的描述。
RM	舍入模式位。参见控制状态寄存器舍入模式位的描述。

控制/状态寄存器条件（CC0）位

当一个浮点比较操作发生时，结果被保存在 CC0 位，即条件位。如果比较结果为真，则 CC0 位被置 1；反之则置 0。CC0 位仅能被浮点比较指令和 CTC1 指令所修改。

控制/状态寄存器导致(Causes)域

控制/状态寄存器的位 17：12 为导致（Causes）域，如图 7-2 所示，这些位反映了最近执行指令的结果。Causes 域是协处理器 0 的 Cause 寄存器的一个逻辑扩充，这些位指示了由上次浮点操作所引起的例外，并且如果相应的使能位（Enable）被设置的话则产生一个中断或者例外。如果一条指令中产生不只一个例外，每一个相应的例外导致位都

要被设置。

Causes 域能被每条浮点操作指令所重写（不包括 Load、Store、Move 操作）。其中如果需要软件仿真来完成的则把该操作的未实现操作位（E）置 1，否则保持为 0。其它位则依照 IEEE754 标准看是否相应的例外产生而分别置 1 或者置 0。

当一个浮点例外发生，没有结果将被存储，状态唯一受影响的就是 Causes 域。

控制/状态寄存器使能(Enables)域

任何时候当 Cause 位和相应的使能位（Enable）同时为 1 时，会产生一个浮点例外。如果浮点操作设置了一个被允许激活（相应使能位为 1）的 Cause 位，则处理器会立即产生一个例外，这和用 CTC1 指令同时设置 Cause 位和 Enable 位为 1 的效果一样。

对于未实现操作(E)来说没有相应的使能位，如果设置了未实现操作，它总是会产生一个浮点例外。

在从一个浮点例外返回之前，软件首先必须用一个 CTC1 指令来清除被激活了的 Cause 位以防止中断的重复执行。因此，在用户态下的运行的程序永远不会观察到被使能的 Cause 位的值为 1；如果用户态的处理程序需要获得该信息，则 Cause 位的内容必须被传递到其它地方而不是在状态寄存器中。

如果浮点操作只设置未被使能（相应的使能位为 0）的 Cause 位，则没有例外发生，同时 IEEE754 标准定义的默认结果被写回。在这种情况下，前一条浮点指令所引起的例外能够通过读 Causes 域的值来确定。

控制/状态寄存器标志(Flags)域

标志位是累积的，它指示自从上次被明确重置后发生了例外。如果一个 IEEE754 例外被产生，那么相应的 Flag 位被置 1，否则保持不变，因此对于浮点运算来说这些位永不会被清除。但是我们可以通过 CTC1 控制指令写一个新值到状态寄存器中来实现对 Flag 位的设置或清除。

当一个浮点例外发生时，Flag 位并不由硬件来设置；浮点例外的处理软件有责任在调用用户程序之前设置这些位。

控制/状态寄存器的舍入模式(RM)域

控制/状态寄存器中第 0 位和第 1 位组成了舍入模式（RM）域。如表 7-3 中所示，FPU 根据这些位所指定的舍入方式来对所有的浮点运算进行相应的舍入处理。

表 7-3 舍入模式位解码

舍入模式 RM(1:0)	助记符	描述
0	RN	把结果向最接近可表示数的方向舍入，当两个最接近可表示数离结果一样接近时，则向最低位为 0 的那个最接近数方向舍入。
1	RZ	向 0 方向舍入：把结果向与之最接近并且在绝对值上不大于

		它的那个数舍入。
2	RP	向正无穷大方向舍入:把结果向与之最接近并且不小于它的那个数舍入
3	RM	向负无穷大方向舍入：把结果向与之最接近并且不大于它的那个数舍入

7.3 浮点部件指令集概述

所有的 FPU 指令都是 32 位长，以字为边界对齐。龙芯 2F 浮点部件不仅运行由 MIPS 标准定义的浮点指令，还加入了一些特殊的指令，如多媒体和双单精度操作，以此增强龙芯 2F 处理器的整体性能。这些特殊指令或者使用与浮点指令相同的操作码，但是扩展了浮点指令的格式域（FMT）,或者采用其它类型的操作码，如协处理器 2（COP2）的操作码。龙芯 2F 浮点部件指令集可以根据不同的格式被分为以下几组：

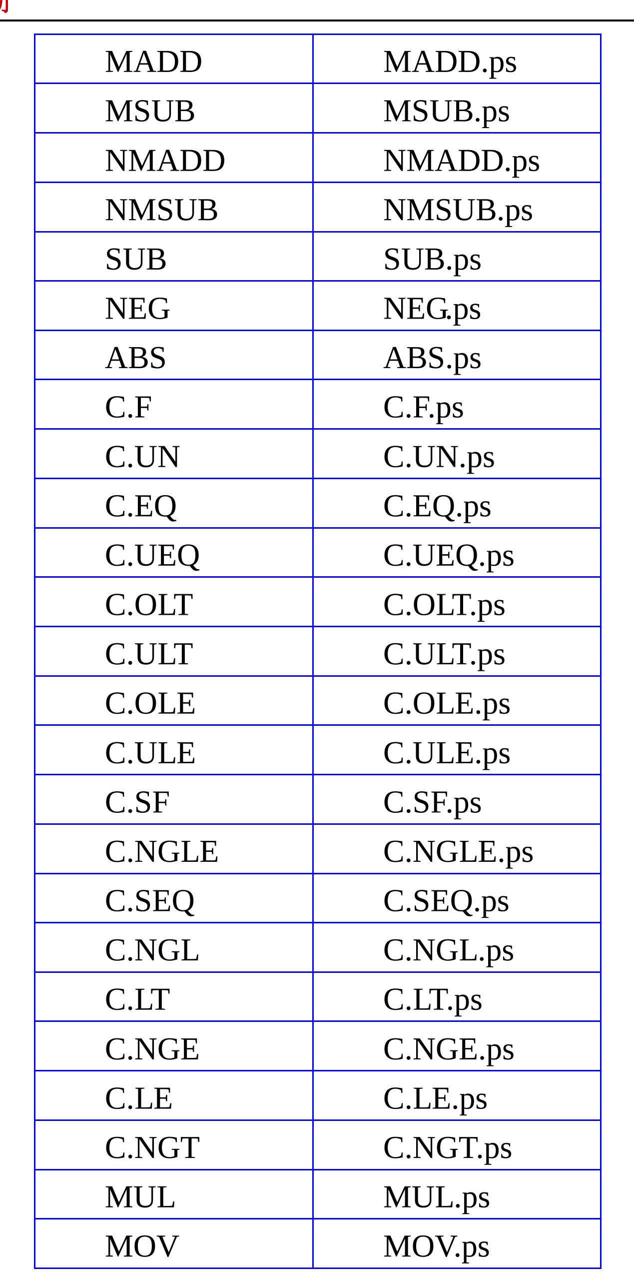
- **单精度或双精度浮点指令 (FMT =16, 17)**。这些指令包括 ADD、SUB、Conversion、Move、Compare 和 Branch 等指令。表 7-4 列出了这些浮点指令
- **双单精度浮点指令(Paired-single，PS) (FMT=22)**。PS 指令可以同时执行两个单精度数的浮点操作，包括 Multiply-ADD、ADD、SUB、MUL、ABS、NEG、Move 和 Compare 运算。表 7-5 详细列出了这些操作。
- **多媒体指令。**龙芯 2F 浮点部件的多媒体指令为增强高级媒体和通讯应用的性能而设计。龙芯 2F 多媒体指令支持基于字节、半字、字以及双字整型数据类型的并行操作。
- **字或双字定点指令。** 这些指令是 MIPS 定点指令的一个子集。它们执行定点操作但分享浮点寄存器和数据通路。在某种意义上它们可被看作多媒体指令的一部分。

表 7-4 龙芯 2F 浮点部件中的浮点指令

MADD	ADD	ROUND.L	MFC1	CVT.S	BC1F	C.F	C.SF
MSUB	SUB	TRUNC.L	MTC1	CVT.D	BC1T	C.UN	C.NGLE
NMADD	MUL	CEIL.L	DMFC1		BC1FL	C.EQ	C.SEQ
NMSUB	DIV	FLOOR.L	DMTC1		BC1TL	C.UEQ	C.NGL
	SQRT	ROUND.W	CFC1	CVT.W		C.OLT	C.LT
	ABS	TRUNC.W	CTC1	CVT.L		C.ULT	C.NGE
	MOV	CEIL.W				C.OLE	C.LE
	NEG	FLOOR.W				C.ULE	C.NGT

表 7-5 龙芯 2F 中的双单精度指令 Paired-single(PS)

Fmt / OP	Fmt=22
ADD	ADD.ps

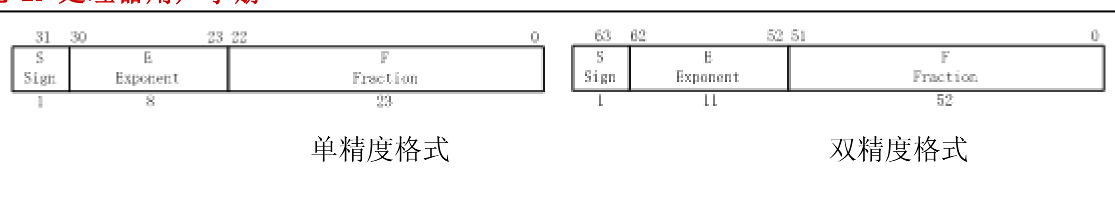


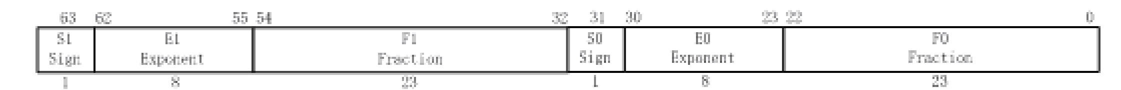
MADD	MADD.ps
MSUB	MSUB.ps
NMADD	NMADD.ps
NMSUB	NMSUB.ps
SUB	SUB.ps
NEG	NEG.ps
ABS	ABS.ps
C.F	C.F.ps
C.UN	C.UN.ps
C.EQ	C.EQ.ps
C.UEQ	C.UEQ.ps
C.OLT	C.OLT.ps
C.ULT	C.ULT.ps
C.OLE	C.OLE.ps
C.ULE	C.ULE.ps
C.SF	C.SF.ps
C.NGLE	C.NGLE.ps
C.SEQ	C.SEQ.ps
C.NGL	C.NGL.ps
C.LT	C.LT.ps
C.NGE	C.NGE.ps
C.LE	C.LE.ps
C.NGT	C.NGT.ps
MUL	MUL.ps
MOV	MOV.ps

7.4 浮点部件格式

7.4.1 浮点格式

FPU 既可以对 32 位（单精度）也可以对 64 位（双精度）符合 IEEE 标准的浮点数进行操作。32 位的单精度格式包括一个 24 比特的以符号－幅度表示的小数域（F＋S）和一个 8 比特的指数域（E）；64 位的双精度格式包括一个 53 比特的符号－幅度表示的小数域（F＋S）和一个 11 比特的指数域（E）；64 位双精度（PS）格式包含两个单精度浮点格式。分别如图 7-3 所示。





双单精度格式

图 7-3 浮点格式

正如上图所示，浮点数的格式由以下三个域组成：

- 符号域，S
- 带偏移的指数域，$E = E_0 + Bias$，E_0 是不带偏移的指数
- 小数域，$F= .b_1b_2....b_{p-1}$

指数 E_0 的范围是包括 E_{min} 和 E_{max} 在内的所有二者之间的整数，另外再加上以下两个保留值：

- $E_{min} - 1$ （用来编码 0 和亚正常数）
- $E_{max} +1$ （用来编码∞和 NaN[Not a Number]）

对于单精度或者双精度格式来说，每一个可表示的非 0 数都有唯一一种编码与之对应。其编码所对应的数值 V 由表 7-6 中的等式所决定。

表 7-6 计算单精度和双精度格式的浮点数的值的公式

NO.	公式
(1)	if $E_0 = E_{max}+1$ and $F\neq0$, then V = NaN, regardless of s
(2)	if $E_0 = E_{max}+1$ and F = 0, then $V= (-1)^S\infty$
(3)	if $E_{min}\leqslant E_0\leqslant E_{max}$, then $V = (-1)^S 2^{E_0} (1.F)$
(4)	if $E_0 = E_{min}-1$ and $F\neq0$, then $V= (-1)^S 2^{Emin}(0.F)$
(5)	if $E_0 = E_{min}-1$ and F = 0, then $V = (-1)^S 0$

对于所有的浮点格式，如果 V 是一个 NaN，那么 F 的最高位决定了这个数是 Signaling NaN 还是 Quiet NaN：如果 F 的最高位被设置，那么 V 是 Signaling NaN，否则 V 是 Quiet NaN。

表 7-7 定义了一些浮点格式的相关参数的值；浮点的最大值和最小值在表 7-8 中给出。

表 7-7 浮点格式参数值

参数	格式	
	单精度	双精度
E_{max}	+127	+1203
E_{min}	-126	-1022
指数偏移量	+127	+1023
指数位宽度	8	11
整数位	隐藏（Hidden）	隐藏（Hidden）
F (小数位宽度)	24	53
格式总宽度	32	64

表 7-8 最大数和最小数的浮点值

类型	值
单精度浮点最小数	1.40129846e–45
单精度浮点最小正规数	1.17549435e–38
单精度浮点最大数	3.40282347e+38
双精度浮点最小数	4.9406564584124654e–324
双精度浮点最小正规数	2.2250738585072014e–308
双精度浮点最大数	1.7976931348623157e+308

7.4.2 多媒体指令格式

多媒体指令为每 64 位数据引入了新的包裹(Packed)数据类型，数据元素可以有如下几类：

- 8 个包裹，每个包裹是一个连续的 8 位字节
- 4 个包裹，每个包裹是一个连续的 16 位半字
- 2 个包裹，每个包裹是一个连续的 32 位字
- 1 个 64 位双字

这 64 位被编号为 0－63。位 0 是最低位（LSB），位 63 是最高位（MSB）。较低位相应存放数据低位，较高位存放数据高位。例如，一个半字包含位 0 到位 15 一共 16 位，如果在一个半字中的一个字节包含位 0 到位 7，那么它在这个半字中称为低字节，如果是包含位 8 到位 15，那么叫做高字节。

包裹的整形数据以两种格式保存，无符号和符号。例如，

图 7-4 说明了包裹的无符号半字格式，图 7-5 说明了包裹的有符号半字格式。

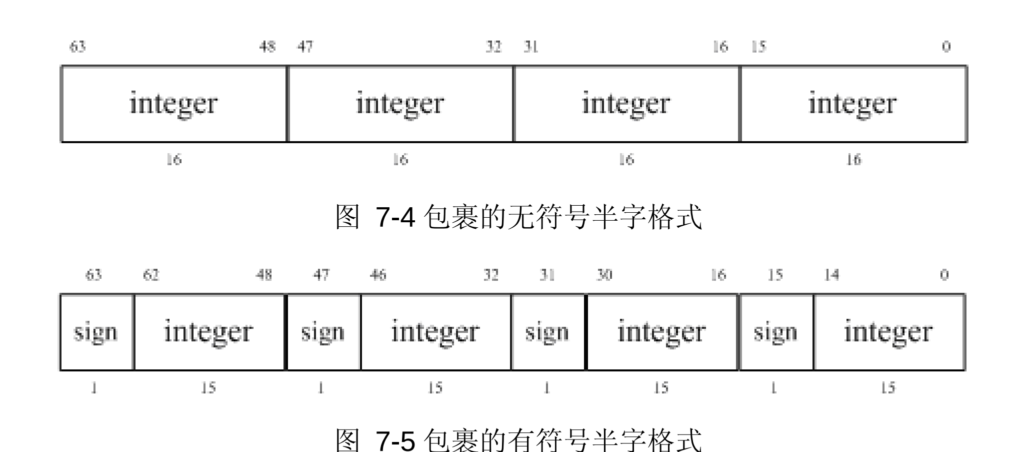


图 7-4 包裹的无符号半字格式

图 7-5 包裹的有符号半字格式

7.5 FPU 指令流水线概述

FPU 提供一个和 CPU 指令流水线并行的指令流水线。它和 CPU 共享基本的 9 级流水线体系结构，但根据浮点操作的不同，执行流水级又细分为 2~6 个流水级。每个 FPU 指令被两个浮点功能单元中的一个执行：FALU1 或者 FALU2。FALU2 单元运行指令操作码为于浮点加减、乘法、乘加、除法以及开平方根等操作的指令； FALU1 单元运行指令操作码为于浮点加减、乘法、乘加以及 FALU2 中所没有的其它浮点操作指令。

每个 FALU 单元每个周期能够接收 1 条指令，并能向浮点寄存器文件送出一个结果。在 FALU1 单元，浮点加减、浮点乘法、浮点乘加运算需要 6 个执行周期；定点与浮点间的格式转换运算需要 4 个执行周期；FALU1 中的其它所有运算（包括 CVT.D.S 和 CVT.S.D）需要 2 个执行周期。这意味着，如果当前的浮点加法指令和下一条浮点指令之间存在着写后读（RAW） 依赖，那么龙芯 2F 在没有对浮点结果进行 Forward 的情况下，下一条指令将等待至少 7 个周期才能被执行；FALU1 单元是完全流水的，所以它不需要给前面的流水级延迟信号。但是可能存在两条有着不同执行周期的指令在同一拍输出结果，在这种情况下，执行周期较短的指令优先向总线输出结果。

FALU2 单元执行浮点加减、浮点乘法、浮点乘加、浮点除法、浮点开平方根操作。其中浮点加减、浮点乘法、浮点乘加操作为全流水操作，需要 6 个执行周期；浮点除法根据操作数的不同，需要 4~16 个执行周期；浮点开平方根操作则需要 4~31 个执行周期。浮点除法和开平方根操作不是流水的，所以如果同时有两个浮点除法指令或者两个浮点开平方根指令在 FALU2 中，那么 FALU2 单元将向前一流水级发出一个停顿信号，并且在除法或开平方根指令写回前不能接收其它指令。

7.6 浮点例外处理

该节描述了浮点计算的例外。浮点例外发生在当 FPU 不能以常规的方式处理操作数或者浮点计算的结果时，FPU 产生相应的例外来启动相应的软件陷阱或者是设置状态标志位。

FPU 的控制和状态寄存器对于每一种例外都包含一个使能位，使能位决定一个例外是否能够导致 FPU 启动一个例外陷阱或者设置一个状态标志。

如果一个陷阱启动，FPU保持操作开始的状态，启动软件例外处理路径；如果没有陷阱启动，一个适当的值写到FPU目标寄存器中，计算继续进行。

FPU 支持五个 IEEE754 例外：

- 不精确 Inexact (I)
- 下溢 Underflow (U)
- 上溢 Overflow (O)
- 除零 Division by Zero (Z)
- 非法操作 Invalid Operation (V)

以及第六个例外：

- 未实现操作 Unimplemented Operation（E）

未实现操作例外用在当 FPU 不能执行标准的 MIPS 浮点结构，包括 FPU 不能决定正确的例外行为的情况。这个例外指示了软件例外处理的执行。未实现操作例外没有使能信号和标志位，当这个例外发生时，一个相应的未实现例外陷阱发生。

IEEE754 的 5 个例外（V，Z，O，U，I）都对应着一个由用户控制的例外陷阱，当 5 个使能位的某一位被设置时，相应的例外陷阱被允许发生。当例外发生时，相应的导致（Cause）位被设置，如果相应的使能（Enable）位没有设置，例外标志（Flag）位被设置。如果使能位被设置，那么标志位不被设置，同时 FPU 产生一个例外给 CPU。随后的例外处理允许该例外陷阱发生。

当没有例外陷阱信号时，浮点处理器采取缺省方式进行处理，提供一个浮点计算例外结果的替代值。不同的例外类型决定了不同的缺省值。表 7-9 列出了 FPU 对于每个 IEEE 例外的默认处理。

表 7-9 例外的默认处理

域	描述	舍入模式	默认操作
I	非精确例外	Any	提供舍入后的结果
U	下溢例外	RN	根据中间结果的符号把结果置 0
		RZ	根据中间结果的符号把结果置 0
		RP	把正下溢修正为最小正数，把负下溢修正为-0
		RM	把负下溢修正为最小负数,把正下溢修正为+0
		RN	根据中间结果的符号把结果置为无穷大

O	上溢例外	RZ	根据中间结果的符号把结果置为最大数
		RP	把负下溢修正为最大负数,把正下溢修正为+∞
		RM	把正下溢修正为最大整数,把负下溢修正为-∞
Z	被0除	Any	提供一个相应的带符号的无穷大数
V	非法操作	Any	提供一个 Quiet Not a Number(QNaN)

下面对导致 FPU 产生每种例外的条件进行了描述，并且详细说明了 FPU 对每个例外导致条件的反应。

不精确例外 (I)

FPU 在发生如下的情况时产生不精确例外：

- 舍入结果非精确
- 舍入结果上溢
- 舍入结果下溢，并且下溢和不精确的使能位都没有被设置，而且 FS 位被设置。

陷阱被使能的结果：如果一个非精确例外陷阱被使能，结果寄存器不被修改，并且源寄存器被保留。因为这种执行模式会影响性能，所以不精确例外陷阱只有在必要的时候才被使能。

陷阱不被使能的结果：如果没有其他软件陷阱发生，舍入或者上溢结果被发送到目标寄存器。

非法操作例外 (V)

当一个可执行的操作的两个操作数或其中的一个操作数是非法时，非法操作例外发出信号通知。如果例外没有陷入，MIPS 定义这个结果是一个 Quiet Not a Number（QNaN）。非法操作包括：

- 加法或者减法：无穷相减。例如：(+∞)+(- ∞)或者(-∞)-(- ∞)
- 乘法：0×∞，对于所有的正数和负数
- 除法：0/0，∞/∞，对于所有的正数和负数
- 当不处理 Unordered 的比较操作的操作数是 Unordered
- 对一个指示信号 NaN 进行浮点比较或者转换
- 任何对 SNaN（Signaling NaN）的数学操作。当其中一个操作数为 SNaN 或者两个都为 SNaN 时会导致这个例外（MOV 操作不被认为是数学操作，但 ABS 和 NEG 被认为是数学操作）
- 开方：$\sqrt{x}$，当 X 小于 0 时

软件可以模拟其他给定源操作数的非法操作的例外。例如在 IEEE754 中利用软件来实现的特定函数：X REM Y，这里当 Y 是 0 或者 X 是无穷的时候；或者当浮点数转化为十进制时发生上溢，是无穷或者是 NaN；或者先验函数例如：ln(5)或者 $\cos^{-1}(3)$。

陷阱被使能的结果：源操作数的值不被发送。

陷阱不使能的结果：如果没有其他例外发生，QNaN被发送到目标寄存器中。

除零例外（Z）

除法运算中当除数是 0 被除数是一个有限的非零的数据时，除零例外发出信号通知。利用软件可以对其他操作产生有符号的无穷值时模拟除零例外，如：ln(0)，sin(π /2)，cos(0)，或者 0^{-1}。

陷阱被使能的情况：结果寄存器不被修改，源寄存器保留。

陷阱不使能的情况：如果没有陷阱发生，结果是有符号的无穷值。

上溢例外（O）

当舍入后的浮点结果的幅度用没有界限的指数来表示时，大于最大的目标模式所表示有限数据，上溢例外发出通知信号。（这个例外同时设置不精确例外和标志位）

陷阱被使能的情况：结果寄存器不被修改，源寄存器保留。

陷阱不使能的情况：如果没有陷阱发生，最后的结果由舍入模式和中间结果的符号来决定。

下溢例外（U）

两个相关的事件导致了下溢例外：

- 一个很小的在 $\pm 2^{Emin}$ 之间的非零结果，由于该结果非常小，因此会导致其后发生下溢例外。
- 用亚正常数据（Denormalized Number）来近似表示这两个小数据所产生的严重的数据失真。

IEEE754 允许用多种不同的方法检测这些事件，但对于所有的操作要求用相同的方法来检测。小数据可以用下面的方法的一种来检测：

- 舍入后（如果一个非零的数据，在指数范围没有界限的情况下来计算，应该严格的位于 $\pm 2^{Emin}$ 之间）
- 舍入前（如果一个非零的数据，在指数和精度范围没有界限的情况下来计算，应该严格的位于 $\pm 2^{Emin}$ 之间）

MIPS 的结构要求微小数据在舍入后检测。精度失真可以用如下方法的一种来检测：

- 亚正常数据的失真（当产生的结果与指数没有界限时计算的结果不同）
- 非精确数据（当产生的结果与指数和精度范围没有界限的情况下计算的结果不同）

MIPS 结构要求精度失真被检测为产生非精确结果。

陷阱被使能的情况：如果下溢或者不精确例外被使能，或者FS位没有设置，产生未实现操作例外，结果寄存器不被修改。

陷阱不使能的情况：如果下溢或者不精确例外不被使能，而且FS位被设置，最后的结果由舍入模式和立即结果的符号位来决定。

未实现操作例外（E）

当执行任何一条为以后定义所保留的操作码或者操作格式指令时，FPU 控制/状态寄存器中的未实现操作导致位被设置并产生陷阱。源操作数和目的寄存器保持不变，同时指令在软件中仿真。IEEE754 中的任何一个例外都能够从仿真操作中产生，这些例外反过来可以被仿真。另外，当硬件不能正确执行一些罕见的操作或者结果条件时，也会产生未实现指令例外。这些包括：

- 亚正常操作数（Denormalized Operand），比较指令除外
- Quite Not a Number 操作数（QNaN），比较指令除外
- 亚正常数据或者下溢，而且当下溢或者不精确使能信号被设置同时 FS 位没有被设置

注意：亚正常和NaN操作只在转换或者计算指令中进入陷阱，在MOV指令中不进入陷阱。

陷阱被使能的情况：原操作数据不被发送。

陷阱不使能的情况：这个陷阱不能被不使能。

8 特权指令

表 8-1 列出了龙芯 2F 处理器特权指令的定义。

表 8-1 龙芯 2F 特权指令

指令	描述
CACHE	CACHE 操作
DMFC0	从 CP0 取双字数据
DMTC0	将双字数据送到 CP0
ERET	例外返回
MFC0	从 CP0 取数据
MTC0	将数据送到 CP0
TLBP	查询 TLB 项
TLBR	用索引读 TLB 表项
TLBWI	用索引填充 TLB 表项
TLBWR	随机填充 TLB 表项

8.1 CP0 传输指令

龙芯 2F 处理器实现的 CP0 传输指令包括 MTC0、MFC0、DMTC0 和 DMFC0。表 8-2 列出了 32/64 位 CP0 寄存器的 CP0 传输指令操作。

表 8-2 CP0 传输指令

指令	CP0 寄存器位数	操作
MFC0 rt, rd	32	rt <- $rd_{31..0}$
MTC0 rt, rd	32	rd <- $rt_{31..0}$
DMFC0 rt, rd	64	rt <- $rd_{63..0}$
DMTC0 rt,rd	64	rd <- $rt_{63..0}$

8.1.1 DMFC0 指令

从 CP0 控制寄存器取双字指令

31 26	25 21	20 16	15 11	10 0
COP0 010000	DMF 00001	rt	rd	0 00000000000
6	5	5	5	11

- **指令格式**

  DMFC0 rt, rd

- **指令描述**

  将 CP0 寄存器 rd 的内容取进通用寄存器 rt。这个操作定义在龙芯 2F 的核心态执行。在用户态或超级用户态执行这条指令会引起协处理器不可用例外。作为目的寄存器的通用寄存器的全部 64 位都写入作为源寄存器的 CP0 寄存器的内容。

- **指令操作**

  GPR[rt] <- CPR[rd]

- **引起的例外**

  协处理器不可用例外

8.1.2 **DMTC0 指令**

将双字送到 CP0 控制寄存器指令

31 26	25 21	20 16	15 11	10 6 5 0
COP0 **010000**	**DMT** **00101**	**rt**	**rd**	**0** **00000000000**
6	5	5	5	11

- **指令格式**

  DMTC0 rt, rd

- **指令描述**

  将通用寄存器 rt 的内容送入 CP0 寄存器 rd。这个操作定义在龙芯 2F 的核心态执行。在用户态或超级用户态执行这条指令会引起协处理器不可用例外。作为目的寄存器的 CP0 寄存器的全部 64 位都写入作为源寄存器的通用寄存器的内容。

- **指令操作**

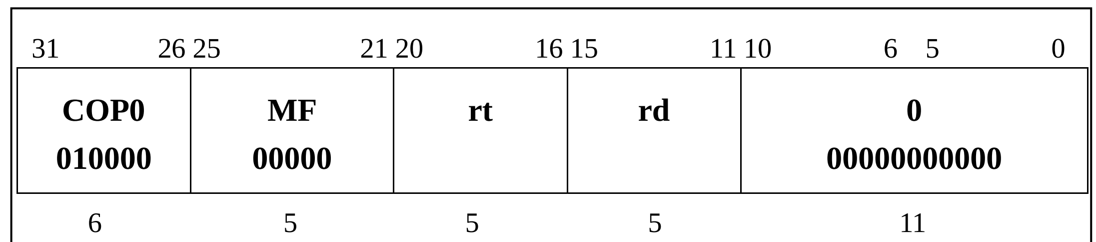
  CPR[rd] <- GPR[rt]

- **引起的例外**

  协处理器不可用例外

8.1.3 **MFC0 指令**

从 CP0 控制寄存器取数据指令



31 26	25 21	20 16	15 11	10 6 5 0
COP0 010000	**MF 00000**	**rt**	**rd**	**0 00000000000**
6	5	5	5	11

- **指令格式**

  MFC0 rt, rd

- **指令描述**

  将 CP0 寄存器 rd 的内容取进通用寄存器 rt。

- **指令操作**

  GPR[rt] <- CPR[rd]

- **引起的例外**

  协处理器不可用例外

8.1.4 MTC0 指令

将数据送到 CP0 控制寄存器指令

31 26	25 21	20 16	15 11	10 6 5 0
COP0 010000	**MT 00100**	**rt**	**rd**	**0 00000000000**
6	5	5	5	11

- **指令格式**

  MTC0 rt, rd

- **指令描述**

  将通用寄存器 rt 的内容送入 CP0 寄存器 rd。

- **指令操作**

  CPR[rd] <- GPR[rt]

- **引起的例外**

  协处理器不可用例外

8.1.5 用户态可用的 CP0 传输指令

为了方便用户程序获得处理器的性能信息，龙芯 2F 处理器允许用户使用 MFC0 和 DMFC0 两条指令访问第 24 号和第 25 号控制寄存器，不会引起协处理器 0 不可用例外。

8.2 TLB 控制指令

龙芯 2F 处理器实现的 TLB 控制指令包括 TLBP、TLBI、TLBWI 和 TLBWR。

8.2.1 TLBP 指令

查询 TLB 表项

31　　26	25	24　　6	5　　0
COP0 **010000**	**CO** **1**	**0** **0000000000000000000**	**TLBP** **001000**
6	1	19	6

- **指令格式**

  TLBP

- **指令描述**

  将内容与 EntryHi 寄存器内容相同的 TLB 表项的地址送入 Index 寄存器。如果没有 TLB 表项与 EntryHi 寄存器的内容相同，Index 寄存器的最高位置为 1。

- **指令操作**

  Index<-1||0^{25}||undefined^{6}

  for I in 0..TLBEntries-1

  if(TLB[i]$_{167..141}$and not(0^{15}||TLB[i]$_{216..205}$))

  =(EntryHi$_{39..13}$ and not(0^{15}||TLB[i]$_{216..205}$)) and

  (TLB[i]$_{140}$ or (TLB[i]$_{135..128}$=EntryHi$_{7..0}$)) then

  　　Index<=0^{26}||i$_{5..0}$

  endif

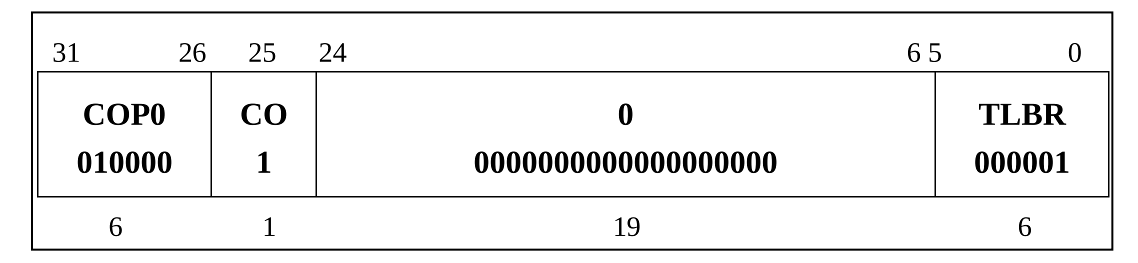
  endfor

- **引起的例外**

  协处理器不可用例外

8.2.2 TLBR 指令

按索引读 TLB 表项



31　　26	25	24　　6	5　　0
COP0 **010000**	**CO** **1**	**0** **0000000000000000000**	**TLBR** **000001**
6	1	19	6

- **指令格式**

  TLBR

- **指令描述**

  将从 TLB 中读出的 G 位（控制 ASID 匹配）写入 EntryLo0 和 EntryLo1 寄存器。将 Index 寄存器索引的 TLB 表项的内容送入 EntryHi 和 EntryLo 寄存器。如果 Index

寄存器的值大于处理器中的 TLB 表项数，则该操作无效。

- **指令操作**

PageMask<-TLB[Index$_{5..0}$]$_{223..192}$

EntryHi<- TLB[Index$_{5..0}$]$_{191..128}$ and not TLB[Index$_{5..0}$]$_{255..192}$

EntryLo1<- TLB[Index$_{5..0}$]$_{127..65}$|| TLB[Index$_{5..0}$]$_{140}$

EntryLo0<- TLB[Index$_{5..0}$]$_{63..1}$|| TLB[Index$_{5..0}$]$_{140}$

- **引起的例外**

协处理器不可用例外

8.2.3 TLBWI 指令

按索引填充 TLB 表项

31 26	25	24 6	5 0
COP0 **010000**	**CO** **1**	**0** **0000000000000000000**	**TLBWI** **000010**
6	1	19	6

- **指令格式**

TLBWI

- **指令描述**

用 EntryLo0 和 EntryLo1 寄存器的 G 位相与的值设置 TLB 的 G 位。用 EntryHi 和 EntryLo 寄存器的内容设置 Index 索引的 TLB 表项。如果 TLB Index 寄存器的值大于处理器中的 TLB 表项数，则该操作无效。

- **指令操作**

TLB[Index$_{5..0}$]<-PageMask||(EntryHi and not PageMask)||EntryLo1||EntryLo0

- **引起的例外**

协处理器不可用例外

8.2.4 TLBWR 指令

随机填充 TLB 表项

31 26	25	24 6	5 0
COP0 **010000**	**CO** **1**	**0** **0000000000000000000**	**TLBWR** **000110**
6	1	19	6

- **指令格式**

TLBWR



- **指令描述**

用 EntryLo0 和 EntryLo1 寄存器的 G 位相与的值设置 TLB 的 G 位。用 EntryHi 和 EntryLo 寄存器的内容设置 TLB Random 寄存器索引的 TLB 表项。

- **指令操作**

TLB[$Random_{5..0}$]<-PageMask||(EntryHi and not PageMask)||EntryLo1||EntryLo0

- **引起的例外**

协处理器不可用例外

8.2.5 ERET 指令

例外返回指令

31　　26	25	24　　　　6	5　　0
COP0 **010000**	**CO** **1**	**0** **0000000000000000000**	**ERET** **011000**
6	1	19	6

- **指令格式**

ERET

- **指令描述**

ERET 指令用来从中断、例外和错误陷入返回。与 Branch 和 Jump 指令不同，ERET 指令不执行下一条指令。ERET 不能作为延迟槽指令。如果处理器在为错误陷入服务（SR_2=1），则从 EPC 中取 PC 值，并且清 Status 寄存器（SR_2）的 ERL 位。否则（SR_2=0），从 EPC 寄存器中取 PC 值，并且清 Status 寄存器（SR_1）的 EXL 位。如果 ERET 指令在 LL 和 SC 指令之间执行，会导致 SC 失败。如果没有例外（Status 寄存器的 EXL=0 并且 ERL=0），将 EXL 置为 0，并跳到 EPC 寄存器保存的地址处。

- **指令操作**

If SR_2=1 then

PC<-ErrorEPC

SR<-$SR_{31..3}$||0||$SR_{1..0}$

else

PC<-EPC

SR<-$SR_{31..2}$||0||SR_0

Endif

LLbit<-0

- **引起的例外**

协处理器不可用例外

8.2.6 CACHE 指令

31 26	25 21	20 16	15 0
COP0 010000	base	op	offset
6	5	5	16

- **指令格式**

CACHE op, offset(base)

- **指令描述**

用符号扩展的 16 位 offset 加上通用 base 寄存器的值形成 Cache 操作的虚地址（VA）。虚地址（VA）通过 TLB 转换成物理地址（PA），5 位的操作码（如表 8-3 所示）指定对这个地址的 Cache 操作。除下表以外的其他 Cache 指令未定义，对 Uncached 地址空间的 Cache 指令操作未定义。

表 8-3 CACHE 指令

Op 域	描述	目标 Cache
00000	Index Invalidate	(I)
00001	Index WriteBack Invalidate	(D)
00101	Index Load Tag	(D)
01001	Index Store Tag	(D)
10001	Hit Invalidate	(D)
10101	Hit WriteBack Invalidate	(D)
11001	Index Load Data	(D)
11101	Index Store Data	(D)
00011	Index WriteBack Invalidate	(S)
00111	Index Load Tag	(S)
01011	Index Store Tag	(S)
10011	Hit Invalidate	(S)
10111	Hit WriteBack Invalidate	(S)
11011	Index Load Data	(S)
11111	Index Store Data	(S)

- **指令操作：**

vAddr<-$((\text{offset}_{15})^{48} \| \text{offset}_{15..0})$+GPR[base]

(pAddr)<-AddressTranslation(vAddr)

CacheOp(op,vAddr,pAddr)

- **引起的例外**

  协处理器不可用例外

Index Invalidate (I)

Index Invalidate（I）指令将指令 Cache 中的对应行的四路的 Cache 块都置为 Invalid 状态。VA[13:5]定义行地址。无效即将指令 Cache 状态位置为 0（Invalid）。

Index WriteBack Invalidate (D)

Index WriteBack Invalidate（D）指令将数据 Cache 中的对应块置为 Invalid 状态。VA[13:5]定义地址，VA[1:0]定义无效的组号。无效即数据 Cache 状态位置为 00（Invalid）。如果数据 Cache 对应块的数据是脏的，则将数据写入二级 Cache。

Index WriteBack Invalidate (S)

Index WriteBack Invalidate（S）指令将二级 Cache 中的对应块置为 Invalid 状态。如果二级 Cache 对应块的数据是脏的，则将数据写到处理器系统接口部件。由于二级 Cache 与数据 Cache 和指令 Cache 保持包含关系，因此在二级 Cache 无效写回前先将数据 Cache 和指令 Cache 中的对应数据无效，如果数据 Cache 中对应的数据是脏的，则先将其写入二级 Cache，最后完成二级 Cache 块的无效写回。PA[16:5]定义物理地址，PA[1:0]定义无效的组号。

无效写回操作过程如下：

1. 处理器从二级 Cache 的 Tag 数组读取 STag 和状态位．如果状态位 State=00（Invalid），则不需要采取任何操作。如果对应 Cache 块是有效的，STag 用来对对指令和数据 Cache 进行操作。
2. 查询指令 Cache，如果指令 Cache 的 ITag=STag 并且指令 Cache 中该块的状态 IState=1（Valid），处理器将指令 Cache 中的对应块无效，即将状态置位为 0（Invalid）。
3. 查询数据 Cache，如果数据 Cache 的 DTag=STag 并且数据 Cache 中该块的状态 DState 不等于 00（Invalid），若 Dirty 位的值为 1，则将数据写入二级 Cache，无效对应 Cache 块。若 Dirty 位的值为 0，则直接无效数据 Cache 的对应块。
4. 将二级 Cache 块的状态置为 00（Invalid）。如果二级 Cache 的状态为 11（Dirty），将对应块写回到处理器接口。

Index Load Tag (D)

Index Load Tag(D)将数据 Cache 的 Tag 域写入 CP0 的 TagLo 和 TagHi 寄存器。VA[13:5]定义地址，VA[1:0]定义读 Tag 的组号。

操作的映射定义如下：

TagLo[5:4] = SCWay

TagLo[7:6] = State bits

TagLo[31:8] = Tag[35:12]

TagHi[3:0] = Tag[39:36]

TagHi[31:29] = StateMod Bits

所有其他 CP0 TagLo 和 TagHi 寄存器位置为 0。

Index Load Tag (S)

Index Load Tag(S)将二级 Cache 的 Tag 域写入 CP0 的 TagLo 和 TagHi 寄存器。PA[16:5]定义地址，PA[1:0]定义读 Tag 的组号。

操作的映射定义如下：

TagLo[11:10] = State Bits

TagLo[31:13] = Tag[35:17]

TagHi[3:0] = Tag[39:36]

所有其他 CP0 TagLo 和 TagHi 寄存器位置为 0。

Index Store Tag (D)

Index Store Tag(D)将 CP0 的 TagLo 和 TagHi 寄存器的值存入数据 Cache 的 Tag 域。VA[13:5]定义地址，VA[1:0]定义读 Tag 的组号。

操作的映射定义如下：

SCWay = TagLo[5:4]

State Bits = TagLo[7:6]

Tag[35:12] = TagLo[31:8]

Tag[39:36] = TagHi[3:0]

Index Store Tag (S)

Index Store Tag(S)将 CP0 的 TagLo 和 TagHi 寄存器的值存入二级 Cache 的 Tag 域。PA[13:5]定义地址，PA[1:0]定义读 Tag 的组号。

操作的映射定义如下：

State Bits = TagLo[11:10]

Tag[35:17] = TagLo[31:13]

Tag[39:36] = TagHi[3:0]

Hit Invalidate (D)

Hit Invalidate (D)无效数据 Cache 匹配地址 PA 的项。VA[13:5]索引的所有的路都从指令 Cache 中读出。

如果 DState 的值不等于 00（Invalid），并且 Cache 指令的 PA 与从数据 Cache 中读出的 DTag 匹配，将该项的 DState 置为 00（Invalid）。

Hit Invalidate (S)

Hit Invalidate（S）指令将二级 Cache 中地址匹配的对应块置为 Invalid 状态。由于二级 Cache 与数据 Cache 和指令 Cache 保持包含关系，因此在二级 Cache 无效写回前先将数据 Cache 和指令 Cache 中的对应数据无效，最后完成二级 Cache 块的无效。

无效操作过程如下：

1. 处理器用 PA 从二级 Cache 的 Tag 数组读取 STag 和状态位。如果 STag 的值与 PA 对应位的值相同，且状态位 State 不等于 00（Invalid），则发生了命中。如果没有发生命中，该 Cache 指令操作完成。
2. 查询指令 Cache，如果指令 Cache 的 PA 与 STag 匹配，并且指令 Cache 中该块的状态 IState=1（Valid），处理器将指令 Cache 中的对应块无效，即将状态置位为 0（Invalid）。
3. 查询数据 Cache，如果数据 Cache 的 DTag=STag 并且数据 Cache 中该块的状态 DState 不等于 00（Invalid），则无效数据 Cache 的对应块。
4. 将二级 Cache 块的状态置为 00（Invalid）。

Hit WriteBack Invalidate (D)

Hit WriteBack Invalidate（D）指令将数据 Cache 中与 PA 地址相匹配的块置为 Invalid 状态。数据 Cache 中的四路都用 VA[13:5]做为索引读出。如果 DState 不等于 00（Invalid），并且 DTag 与 Cache 指令的 PA 相匹配，则将数据 Cache 的该项状态位置为 00（Invalid）。如果数据 Cache 对应块的数据是脏的，则将数据写入二级 Cache。

Hit WriteBack Invalidate (S)

Hit WriteBack Invalidate（S）指令二级 Cache 中地址 PA 匹配的对应块置为 Invalid 状态。如果二级 Cache 对应块的数据是脏的，则将数据写到处理器系统接口部件。由于二级 Cache 与数据 Cache 和指令 Cache 保持包含关系，因此在二级 Cache 无效写回前先将数据 Cache 和指令 Cache 中的对应数据无效，如果数据 Cache 中对应的数据是脏的，则先将其写入二级 Cache，最后完成二级 Cache 块的无效写回。

无效写回操作过程如下：

1. 处理器用 PA 从二级 Cache 的 Tag 数组读取 STag 和状态位。如果 STag 的值与 PA 对应位的值相同，且状态位 State 不等于 00（Invalid），则发生了命中。如果没有发生命中，该 Cache 指令操作完成。
2. 查询指令 Cache，如果指令 Cache 的 ITag=STag 并且指令 Cache 中该块的状态 IState=1（Valid），处理器将指令 Cache 中的对应块无效，即将状态置位为 0（Invalid）。
3. 查询数据 Cache，如果数据 Cache 的 DTag=STag 并且数据 Cache 中该块的状态 DState 不等于 00（Invalid），若 Dirty 位的值为 1，则将数据写入二级 Cache，无效对应 Cache 块。若 Dirty 位的值为 0，则直接无效数据 Cache 的对应块。
4. 将二级 Cache 块的状态置为 00（Invalid）。如果二级 Cache 的状态为 11（Dirty），将对应块写回到处理器接口。

Index Load Data (D)

Index Load Data（D）将数据 Cache 的 Data 域数据存入 TagHi 和 TagLo 寄存器。VA[13:3]

为地址，VA[1:0]指示对应的组号。

Index Load Data (S)

Index Load Data（S）将二级 Cache 的 Data 域数据存入 TagHi 和 TagLo 寄存器。PA[16:3]为地址，PA[1:0]指示对应的组号。

Index Store Data (D)

Index Store Data（D）将 TagHi 和 TagLo 寄存器的数据存入数据 Cache 的 Data 域。VA[13:3]为地址，VA[1:0]指示对应的组号。

Index Store Data (S)

Index Store Data（S）将 TagHi 和 TagLo 寄存器的数据存入二级 Cache 的 Data 域。PA[16:3]为地址，PA[1:0]指示对应的组号。

9 地址窗口配置模块

在 2F 中有 CPU 地址空间、DDR2 地址空间、以及 PCI 地址空间共三个 IP 相关的地址空间。地址窗口是供 CPU 和 PCI-DMA 两个具有 master 功能的 IP 进行路由选择和地址转换而设置的。CPU 和 PCIDMA 两者都拥有 4 个地址窗口，可以完成目标地址空间的选择以及从源地址空间到目标地址空间的转换。每个地址窗口由 BASE、MASK 和 MMAP 三个 64 位寄存器组成，BASE 以 M 字节对齐，MASK 采用类似网络掩码高位为 1 的格式，MMAP 的低两位是路由选择，表示对应新地址空间的编号，其中 DDR2 的标号为 0，PCI/Local IO 编号为 1。对这些配置寄存器的赋值通过 64 位的双字写进行。

窗口命中公式:　　(IN_ADDR & MASK) == BASE

新地址换算公式:　　OUT_ADDR = (IN_ADDR & ~MASK) | {MMAP[63:20],20’h0}

根据缺省的寄存器配置，芯片启动后：CPU 的 0x0-0x0fff_ffff 的地址区间（256M）映射到 DDR2 的 0x0-0x0fff_ffff 的地址区间，CPU 的 0x1000_0000-0x1fff_ffff 区间(256M）映射到 PCI 的 0x1000_0000-0x1fff_ffff 区间 PCIDMA 的 0x8000_0000 到 0x8fff_ffff 的地址区间（256M）映射到 DDR 的 0x0-0x0fff_ffff 的地址区间。软件可以通过修改相应的配置寄存器实现新的地址空间路由和转换。

表 9-1 地址窗口寄存器地址

地址	寄存器	描述	缺省值
3ff0 0000	CPU_WIN0_BASE	CPU 窗口 0 的基地址	0x0
3ff0 0008	CPU_WIN1_BASE	CPU 窗口 1 的基地址	0x1000_0000
3ff0 0010	CPU_WIN2_BASE	CPU 窗口 2 的基地址	0xffff_ffff_fff0_0000
3ff0 0018	CPU_WIN3_BASE	CPU 窗口 3 的基地址	0xffff_ffff_fff0_0000
3ff0 0020	CPU_WIN0_MASK	CPU 窗口 0 的掩码	0xffff_ffff_f000_0000
3ff0 0028	CPU_WIN1_MASK	CPU 窗口 1 的掩码	0xffff_ffff_f000_0000
3ff0 0030	CPU_WIN2_MASK	CPU 窗口 2 的掩码	0xffff_ffff_fff0_0000
3ff0 0038	CPU_WIN3_MASK	CPU 窗口 3 的掩码	0xffff_ffff_fff0_0000
3ff0 0040	CPU_WIN0_MMAP	CPU 窗口 0 的新基地址	0x0
3ff0 0048	CPU_WIN1_MMAP	CPU 窗口 1 的新基地址	0x1000_0001
3ff0 0050	CPU_WIN2_MMAP	CPU 窗口 2 的新基地址	0
3ff0 0058	CPU_WIN3_MMAP	CPU 窗口 3 的新基地址	0
3ff0 0060	PCIDMA_WIN0_BASE	PCIDMA 窗口 0 的基地址	0x8000_0000
3ff0 0068	PCIDMA_WIN1_BASE	PCIDMA 窗口 1 的基地址	0xffff_ffff_fff0_0000
3ff0 0070	PCIDMA_WIN2_BASE	PCIDMA 窗口 2 的基地址	0xffff_ffff_fff0_0000
3ff0 0078	PCIDMA_WIN3_BASE	PCIDMA 窗口 3 的基地址	0xffff_ffff_fff0_0000

3ff0 0080	PCIDMA_WIN0_MASK	PCIDMA 窗口 0 的掩码	0xffff_ffff_fff0_0000
3ff0 0088	PCIDMA_WIN1_MASK	PCIDMA 窗口 1 的掩码	0xffff_ffff_fff0_0000
3ff0 0090	PCIDMA_WIN2_MASK	PCIDMA 窗口 2 的掩码	0xffff_ffff_fff0_0000
3ff0 0098	PCIDMA_WIN3_MASK	PCIDMA 窗口 3 的掩码	0xffff_ffff_fff0_0000
3ff0 00a0	PCIDMA_WIN0_MMAP	PCIDMA 窗口 0 的新基地址	0x0
3ff0 00a8	PCIDMA_WIN1_MMAP	PCIDMA 窗口 1 的新基地址	0xffff_ffff_fff0_0000
3ff0 00b0	PCIDMA_WIN2_MMAP	PCIDMA 窗口 2 的新基地址	0xffff_ffff_fff0_0000
3ff0 00b8	PCIDMA_WIN3_MMAP	PCIDMA 窗口 3 的新基地址	0xffff_ffff_fff0_0000

此外，当出现由于 CPU 猜测执行引起对非法地址的读访问时， 4 个地址窗口都不命中， 由配置寄存器模块返回全 0 的数据给 CPU，以防止 CPU 死等。

10 DDR2 SDRAM 控制器配置

龙芯 2F 处理器内部集成的内存控制器的设计遵守 DDR2 SDRAM 的行业标准（JESD79-2B）。在龙芯 2F 处理器中，所实现的所有内存读/写操作都遵守 JESD79-2B 的规定。

10.1 DDR2 SDRAM 控制器功能概述

龙芯 2F 处理器支持最大 4 个物理内存 bank（由 4 个 DDR2 SDRAM 片选信号实现），一共含有 18 位的地址总线（即：15 位的行列地址总线和 3 位的逻辑 Bank 总线）。最大支持的地址空间为 128GB（2^{37}）。

龙芯 2F 处理器支持 JESD79-2B 标准中所规定的所有内存芯片类型，具体选择使用不同内存芯片类型时，可以调整 DDR2 控制器参数设置进行支持。其中，支持的最大片选（CS_n）数为 4，行地址（RAS_n）数为 15，列地址（CAS_n）数为 14，逻辑体选择（BANK_n）数为 3。

CPU 发送的内存请求物理地址将按照如下图所示的方法进行行列地址转换：

以 4GB 地址空间为例，按照下面的配置：

片选 = 4　　Bank 数 = 8

行地址数 = 12　　列地址数 = 12

36 — 32	31 — 30	29 — 18	17 — 15	14 — 3	2 — 0
	片选	行选	BANK选择	列选	生成字节使能

图 10-1 DDR2 SDRAM 行列地址与 CPU 物理地址的转换

龙芯 2F 处理器所集成的内存控制电路只接受来自处理器或者外部设备的内存读/写请求，在所有的内存读/写操作中，内存控制电路处于从设备状态（Slave State）。

龙芯 2F 处理器内部集成的内存控制器实现了一种动态的 Page 管理策略，针对一次访存操作，内存控制器对 Open Page 策略/Close Page 策略的选择是由硬件电路来实现的，无需软件设计人员来干预。另外，龙芯 2F 处理器中内存控制器具有如下特征：

- 接口上命令、读写数据全流水操作
- 内存命令合并、排序提高整体带宽
- 配置寄存器读写端口，可以修改内存设备的基本参数
- 内建动态延迟补偿电路（DCC），用于数据的可靠发送和接收
- ECC 功能可以对数据通路上的 1 位和 2 位错误进行检测，并能对 1 位错误进行自动纠错
- 支持 133-333MHZ 工作频率

10.2 DDR2 SDRAM 读操作协议

DDR2 SDRAM 读操作的协议如图 10-2 所示。在图中命令（Command，简称 CMD）由 RAS_n，CAS_n 和 WE_n，共三个信号组成。对于读操作，RAS_n=1，CAS_n＝0，WE_n＝1。

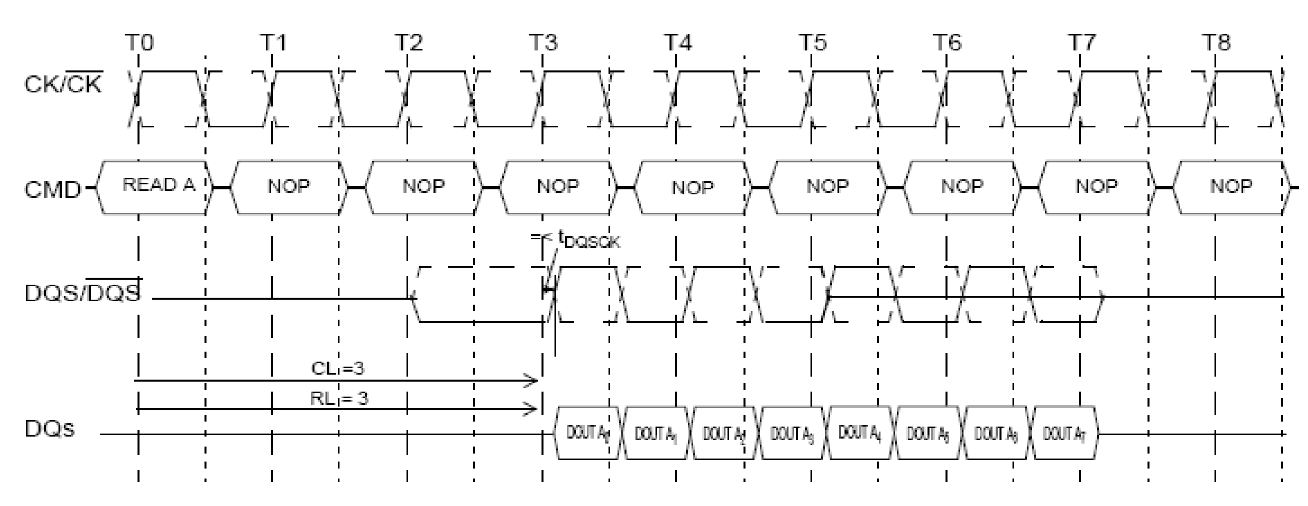


图 10-2 DDR2 SDRAM 读操作协议

上图中，Cas Latency (CL) = 3，Read Latency (RL) = 3，Burst Length = 8。

10.3 DDR2 SDRAM 写操作协议

DDR2 SDRAM 写操作的协议如图 10-3 所示。在图中命令 CMD 是由 RAS_n，CAS_n 和 WE_n，共三个信号组成的。对于写操作，RAS_n=1，CAS_n＝0，WE_n＝0。另外，与读操作不同，写操作需要 DQM 来标识写操作的掩码，即需要写入的字节数。DQM 与图中 DQs 信号同步。

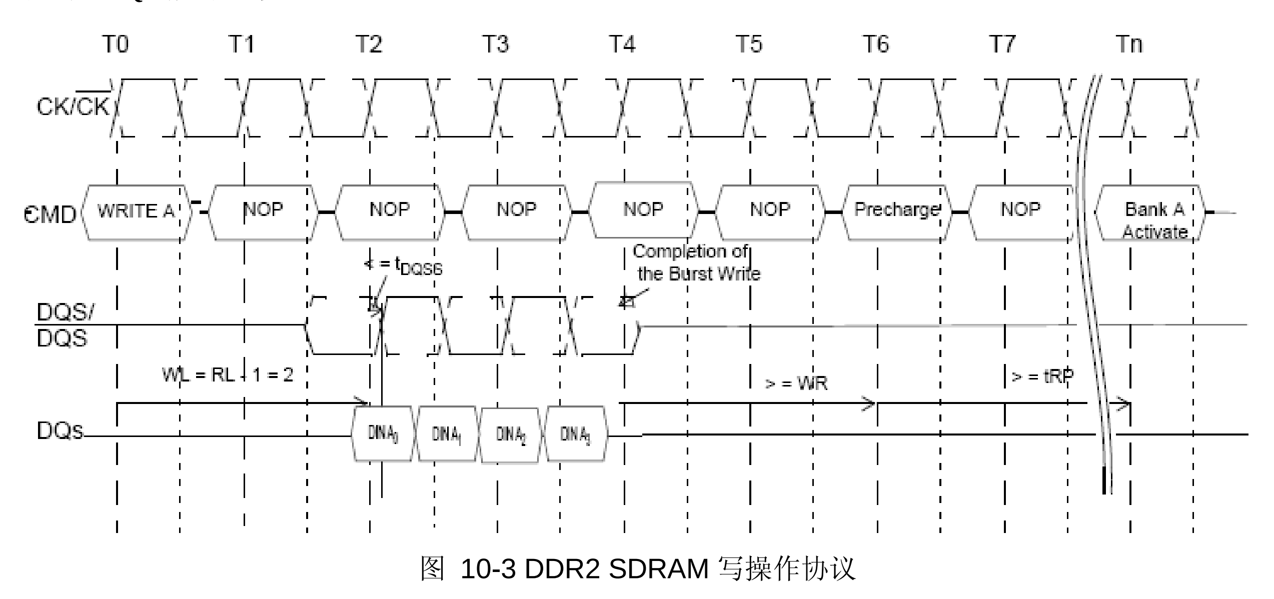


图 10-3 DDR2 SDRAM 写操作协议

上图中，Cas Latency (CL) = 3，Write Latency (WL) = Read Latency (RL) – 1 = 2，Burst Length = 4。

10.4 DDR2 SDRAM 参数配置格式

由于系统中可能使用不同类型的 DDR2 SDRAM，因此，在系统上电复位以后，需要对 DDR2 SDRAM 进行配置。在 JESD79-2B 中规定了详细的配置操作和配置过程，在没有完成 DDR2 的内存初始化操作之前，DDR2 不可用。内存初始化操作执行顺序如下：

(1) 系统复位，aresetn 信号被置 0，此时控制器内部所有寄存器内容将被清除为初始值。

(2) 系统解复位，aresetn 信号被置为 1。

(3) 向配置寄存器地址发 64 位写指令，配置所有 29 个配置寄存器。此时如果写 CTRL_03，应将其中参数 START 设为 0。所有寄存器都必须正确配置才可以正常工作。

(4) 向配置寄存器 CTRL_03 中发 64 位写指令。此时应将参数 START 设为 1。结束后内存控制器将自动对内存发起初始化指令。

在龙芯 2F 处理器设计中，DDR2 SDRAM 的配置在系统主板初始化完成以后，需要使用内存之前，进行内存类型的配置。具体的配置操作是对物理地址 0x0000 0000 0FFF FE00 相对应的 29 个 64 位寄存器写入相应的配置参数。一个寄存器可能会包括一个或多个参数的数据。这些配置寄存器及其包含的参数意义如下表（寄存器中未使用的位均为保留位），表中还给出了基于 DDR2 667 的一种寄存器配置方式，具体的配置可以根据实际情况再决定：

表 10-1 DDR2 SDRAM 配置参数寄存器格式

参数名称	位	缺省值	范围	描述
CONF_CTL_00[31:0] Offset: 0x00				DDR2 667：0x00000101
AREFRESH	24:24	0x0	0x0-0x1	根据 auto_refresh_mode 参数的设置，向内存发起自动刷新命令（只写）
AP	16:16	0x0	0x0-0x1	是否使能内存控制器自动刷新功能，置 1，表示内存访问为 CLOSE PAGE 方式。
ADDR_CMP_EN	8:8	0x0	0x0-0x1	是否允许命令队列重排序逻辑对地址冲突进行检测
ACTIVE_AGING	0:0	0x0	0x0-0x1	是否允许对命令队列中的命令进行 aging 记录，防止低优先级命令饿死
CONF_CTL_00[63:32] Offset: 0x00				DDR2 667：0x01000100
DDR2_SDRAM_MODE	56:56	0x0	0x0-0x1	内存控制器 DDRI 和 DDRII 模式设置，对于 DDRII，应当置 1
CONCURRENTAP	48:48	0x0	0x0-0x1	是否允许控制器对一个 bank 进行 auto precharge 时，对另外一个 bank 发出命令。注：部分内存条不支持

BANK_SPLIT_EN	40:40	0x0	0x0-0x1	是否允许命令队列重排序逻辑对 bank 进行拆分（split）
AUTO_REFRESH_MODE	32:32	0x0	0x0-0x1	设置 auto-refresh 是在下一个 burst 还是下一个命令边界发出
CONF_CTL_01[31:0]　Offset: 0x10				DDR2 667：0x00010000
ECC_DISBALE_W_UC_ERR	24:24	0x0	0x0-0x1	当检测到不可恢复的错误时，是否将 ECC 关闭
DQS_N_EN	16:16	0x0	0x0-0x1	是否使能差分 DQS
DLL_BYPASS_MODE	8:8	0x0	0x0-0x1	是否使能 DLL BYPASS 模式，在 DLL BYPASS 模式下，所有 DLL 的参数将会使用带_BYPASS 结尾的参数，BYPASS 方式与普通方式下，计算的单位是不同的，带 BYPASS 的参数是以延迟线个数为单位，其它的参数是以周期的 1/128 为单位。通常情况下，不需要设置为 DLL_BYPASS_MODE。
DLLLOCKREG	0:0	0x0	0x0-0x1	指示 DLL 是否已锁定（只读），只有在 DLL 锁定之后，对内存发起的读写操作才能有效到达内存，所以，可以用本位判断第一次写内存的时机。
CONF_CTL_01[63:32]　Offset: 0x10				DDR2 667：0x00000000
FWC	56:56	0x0	0x0-0x1	是否强制进行写检查，当这个参数设置后，内存控制器将用 xor_check_bits 参数指定的数与数据进行异或写入内存（只写）
FAST_WRITE	48:48	0x0	0x0-0x1	是否允许控制器打开快速写功能。打开快速写功能后，控制器在未收到全部写数据后即向内存模块发出写命令。
ENABLE_QUICK_SREFRESH	40:40	0x0	0x0-0x1	是否使能快速自刷新。当这个参数使能后，内存的初始化未进行完就进入自刷新状态
EIGHT_BANK_MODE	32:32	0x0	0x0-0x1	指示内存模块是否有 8 个 bank
CONF_CTL_02[31:0]　Offset: 0x20				DDR2 667：0x00000000
NO_CMD_INIT	24:24	0x0	0x0-0x1	在内存初始化过程中，是否禁止在内存模块的 tDLL 时间内发出其它命令
INTRPTWRITENA	16:16	0x0	0x0-0x1	是否允许用 autoprecharge 命令加上对同一 bank 的其它写命令打断前一个写命令
INTRPTREADA	8:8	0x0	0x0-0x1	是否允许用 autoprecharge 命令加上对同一 bank 的其它读命令打断前一个读命令
INTRPTAPBURST	0:0	0x0	0x0-0x1	是否允许对另一 bank 的其它命令打断当前的 auto-precharge 命令
CONF_CTL_02[63:32]　Offset: 0x20				DDR2 667：0x01000101

PRIORITY_EN	56:56	0x0	0x0-0x1	是否使能命令队列重排序逻辑使用优先级
POWER_DOWN	48:48	0x0	0x0-0x1	当使能这个参数时，内存控制器将用 pre-charge 命令关闭内存模块的所有页面，使时钟使能信号为低，不发送收到的所有命令，直到这个参数重新设置为 0
PLACEMENT_EN	40:40	0x0	0x0-0x1	是否使能命令重排序逻辑
ODT_ADD_TURN_CLK_EN	32:32	0x0	0x0-0x1	在对不同片选的快速背对背读或者写命令中间是否插入一个 turn-around 时钟。通常情况下，插入一个这样的周期是对内存是需要的。
CONF_CTL_03[31:0] Offset: 0x30				DDR2 667：0x01000000
RW_SAME_EN	24:24	0x0	0x0-0x1	在命令队列重排序逻辑中是否考虑对同一 bank 读写命令的重组
REG_DIMM_EN	16:16	0x0	0x0-0x1	是否使能 registered DIMM 内存模组
REDUC	8:8	0x0	0x0-0x1	是否只使用 32 位位宽的内存数据通道，通常情况下，不应设置该位
PWRUP_SREFRESH_EXIT	0:0	0x0	0x0-0x1	是用 self-refresh 命令而不是用正常的内存初始化命令来脱离下电模式
CONF_CTL_03[63:32] Offset: 0x30				DDR2 667：0x01010000
SWAP_PORT_RW_SAME_EN	56:56	0x0	0x0-0x1	当 swap_en 使能时，该参数决定是否将同一端口上的类似命令进行交换
SWAP_EN	48:48	0x0	0x0-0x1	在使能命令队列重排序逻辑时，当高优先级命令到达时，是否将正在执行的命令与新命令交换
START	40:40	0x0	0x0-0x1	是否开始内存的初始化工作。需要在所有的参数配置完成之后，再设置该位，让内存进入初始化配置。在没有完成其它位的配置之前就配置该位，很可能导致内存访问错误。
SREFRESH	32:32	0x0	0x0-0x1	内存模块是否进入自刷新工作模式
CONF_CTL_04[31:0] Offset: 0x40				DDR2 667：0x00010101
WRITE_MODEREG	24:24	0x0	0x0-0x1	是否写内存模块的 EMRS 寄存器（只写）
WRITEINTERP	16:16	0x0	0x0-0x1	定义是否能用一个读命令取打断一个写突发
TREF_ENABLE	8:8	0x0	0x0-0x1	是否使能控制器内部的自动刷新功能，通常的情况下，应该将该位置 1
TRAS_LOCKOUT	0:0	0x0	0x0-0x1	是否在 tRAS 时间到期之前发出 auto-prechareg 命令
CONF_CTL_04[63:32] Offset: 0x40				DDR2 667：0x01000202
RTT_0	57:56	0x0	0x0-0x3	定义所有内存模块的片上终端电阻的阻值。这个值将在向内存发起初始化操作时写入内存颗粒。具体的配

				置应当参考相应的内存颗粒手册。 00 –disable 01 – 75ohm 10 – 150 ohm 11 -reserved
CTRL_RAW	49:48	0x0	0x0-0x3	设置 ECC 的检错和纠错模式 2'b00 – 不使用 ECC 2'b01 – 只报错，不纠错 2'b10 – 没有使用 ECC 设备 2'b11 – 使用 ECC 报错纠错
AXI0_W_PRIORITY	41:40	0x0	0x0-0x3	设置 AXI0 端口写命令优先级
AXI0_R_PRIORITY	33:32	0x0	0x0-0x3	设置 AXI0 端口读命令优先级
CONF_CTL_05[31:0]	Offset: 0x50			DDR2 667：0x04050202
COLUMN_SIZE	26:24	0x0	0x0-0x7	设置实际列地址数和最大列地址数(14)之间的差值，应该根据具体的内存颗粒进行配置。 内存所用列地址数 = 14 - COLUMN_SIZE
CASLAT	18:16	0x0	0x0-0x7	设置 CAS latency 值。应当根据具体的内存颗粒在不同的运行频率下进行配置。
ADDR_PINS	10:8	0x0	0x0-0x7	设置实际地址引脚数和最大地址数(15)之间的差值 内存所用地址线数 = 14 – ADDR_PINS
RTT_PAD_TERMINATION	1:0	0x0	0x0-0x3	设置内存控制器 pad 的终端电阻阻值，与内存颗粒上的终端电阻相对应，具体的配置应当参考相应的内存颗粒手册。
CONF_CTL_05[63:32]	Offset: 0x50			DDR2 667：0x00000000
Q_FULLNESS	58:56	0x0	0x0-0x7	定义内存控制器命令队列中有多少命令时认为命令队列满
PORT_DATA_ERROR_TYPE	50:48	0x0	0x0-0x7	定义内存控制器端口上数据错误类型（只读） 位 0 – 突发数据个数大于 16 位 1 – 写数据交错 位 2 – ECC 2 位错
OUT_OF_RANGE_TYPE	42:40	0x0	0x0-0x7	定义发生越界访问时的错误类型（只读）
MAX_CS_REG	34:32	0x4	0x0-0x4	定义控制器所用片选个数（只读）
CONF_CTL_06[31:0]	Offset: 0x60			DDR2 667：0x03050203
TRTP	26:24	0x0	0x0-0x7	定义内存模组的读命令到 precharge 周期数，需要根据具体内存颗粒及运行频率进行配置。
TRRD	18:16	0x0	0x0-0x7	定义到不同 bank 的 active 命令时间间隔，需要根据具

				体内存颗粒及运行频率进行配置。
TEMRS	10:8	0x0	0x0-0x7	定义内存模组初始化时的 emrs 时间，一般内存颗粒配置的周期为 2。
TCKE	2:0	0x0	0x0-0x7	定义 CKE 信号最小脉宽
CONF_CTL_06[63:32]	Offset: 0x60			DDR2 667：0x0a040306
APREBIT	59:56	0x0	0x0-0xf	定义用哪位地址线向内存发出 autoprecharge 命令，一般为 bit 10。
WRLAT	50:48	0x0	0x0-0x7	定义从写命令发出到接收到第一个数据的时间（按时钟周期数），一般为 CASLAT -1。
TWTR	42:40	0x0	0x0-0x7	定义从写命令切换到读命令所需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。
TWR_INT	34:32	0x0	0x0-0x7	定义内存模组的写恢复时间，需要根据具体内存颗粒及运行频率进行配置。
CONF_CTL_07[31:0]	Offset: 0x70			DDR2 667：0x000f0a0b
ECC_C_ID	27:24	0x0	0x0-0xf	发生 1bit ECC 错的源 ID 号（只读）
CS_MAP	19:16	0x0	0x0-0xf	定义可用片选信号，本参数应当根据实际使用的片选个数进行正确的配置，不正确的配置将会导致错误的内存访问。该参数的四位分别对应于 CS0- CS3
CASLAT_LIN_GATE	11:8	0x0	0x0-0xf	定义读命令返回数据时的 gate open 信号打开的时间，一般等于 CASLAT_LIN（以半个时钟周期为单位）
CASLAT_LIN	3:0	0x0	0x0-0xf	当板上走线延迟为 DDR2 时钟周期的 0.5~1.5 倍：CASLAT_LIN = CASLAT×2 小于 0.5 倍：CASLAT_LIN = CASLAT×2-1 大于 1.5 倍：CASLAT_LIN = CASLAT×2+1 （以半个时钟周期为单位）
CONF_CTL_07[63:32]	Offset: 0x70			DDR2 667：0x00000400
MAX_ROW_REG	59:56	0xf	0x0-0xf	系统最大行地址个数（只读）
MAX_COL_REG	51:48	0xe	0x0-0xe	系统最大列地址个数（只读）
INITAREF	43:40	0x0	0x0-0xf	定义系统初始化时所需要执行的 autorefresh 命令个数
ECC_U_ID	35:32	0x0	0x0-0xf	定义发生不可纠错的双字节错的源 ID 号（只读）
CONF_CTL_08[31:0]	Offset: 0x80			DDR2 667：0x01020408
ODT_RD_MAP_CS3	27:24	0x0	0x0-0xf	定义 CS3 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-

				CS3
ODT_RD_MAP_CS2	19:16	0x0	0x0-0xf	定义 CS2 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-CS3
ODT_RD_MAP_CS1	11:8	0x0	0x0-0xf	定义 CS1 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-CS3
ODT_RD_MAP_CS0	3:0	0x0	0x0-0xf	定义 CS0 有读命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-CS3
CONF_CTL_08[63:32]	Offset: 0x80			DDR2 667：0x01020408
ODT_WR_MAP_CS3	59:56	0x0	0x0-0xf	定义 CS3 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-CS3
ODT_WR_MAP_CS2	51:48	0x0	0x0-0xf	定义 CS2 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-CS3
ODT_WR_MAP_CS1	43:40	0x0	0x0-0xf	定义 CS1 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-CS3
ODT_WR_MAP_CS0	35:32	0x0	0x0-0xf	定义 CS0 有写命令时，将指定的 CS 的 ODT 终端电阻有效，具体的配置应当参考相应的内存颗粒手册对于 ODT 配置的要求。该参数的四位分别对应于 CS0-CS3
CONF_CTL_09[31:0]	Offset: 0x90			DDR2 667：0x00000000
PORT_DATA_ERROR_ID	27:24	0x0	0x0-0xf	端口上发生数据错误时的 ID 号（只读）
PORT_CMD_ERROR_TYPE	19:16	0x0	0x0-0xf	端口上发生命令错误的类型（只读） 位 0 – 数据位宽过大 位 1 – 关键字优先操作地址未对齐 位 2 – 关键字优先操作字数不是 2 幂

				位 3 – narrow transform 出错
PORT_CMD_ERROR_ID	11:8	0x0	0x0-0xf	端口上发生命令错误的 ID 号（只读）
OUT_OF_RANGE_SOURCE_ID	3:0	0x0	0x0-0xf	端口上发生越界访问错误时的 ID 号（只读）
CONF_CTL_09[63:32]	Offset: 0x90			DDR2 667：0x0000060c
OCD_ADJUST_PUP_CS0	60:56	0x0	0x0-0x1f	设置内存模组片选 0 OCD 上拉调整值。内存控制器将在初始化时根据这个参数的值向内存模组发出 OCD 调整命令
OCD_ADJUST_PDN_CS0	52:48	0x0	0x0-0x1f	设置内存模组片选 0 OCD 下拉调整值。内存控制器将在初始化时根据这个参数的值向内存模组发出 OCD 调整命令
TRP	43:40	0x0	0x0-0xf	定义内存模组执行 pre-charge 所需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。
TDAL	35:32	0x0	0x0-0xf	当 auto-precharge 参数设置后，该参数定义了 auto-precharge 和 write recovery 时钟周期数。 TDAL = auto-precharge + write recovery 该参数仅在设置了 AP 之后才生效。
CONF_CTL_10[31:0]	Offset: 0xa0			DDR2 667：0x3f130200
AGE_COUNT	29:24	0x0	0x0-0x3f	定义命令队列重排序逻辑使用 aging 算法时每个命令的 aging 初始值
TRC	20:16	0x0	0x0-0x1f	定义对内存模组同一 bank 的 active 命令之间的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。
TMRD	12:8	0x0	0x0-0x1f	定义配置内存模组模式寄存器需要的时钟周期数，通常为 2 个周期
TFAW	4:0	0x0	0x0-0x1f	定义内存模组 tFAW 参数，8 个逻辑 bank 时使用
CONF_CTL_10[63:32]	Offset: 0xa0			DDR2 667：0x1515153f
DLL_DQS_DELAY_2	62:56	0x0	0x0-0x7f	定义读数据时 DQS2 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
DLL_DQS_DELAY_1	54:48	0x0	0x0-0x7f	定义读数据时 DQS1 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
DLL_DQS_DELAY_0	46:40	0x0	0x0-0x7f	定义读数据时 DQS0 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
COMMAND_AGE_COUNT	37:32	0x0	0x0-0x3f	定义命令队列重排序逻辑使用 aging 算法时每个命令

				的 aging 初始值
CONF_CTL_11[31:0]	Offset: 0xb0			DDR2 667：0x15151515
DLL_DQS_DELAY_6	30:24	0x0	0x0-0x7f	定义读数据时 DQS6 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
DLL_DQS_DELAY_5	22:16	0x0	0x0-0x7f	定义读数据时 DQS5 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
DLL_DQS_DELAY_4	14:8	0x0	0x0-0x7f	定义读数据时 DQS4 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
DLL_DQS_DELAY_3	6:0	0x0	0x0-0x7f	定义读数据时 DQS3 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
CONF_CTL_11[63:32]	Offset: 0xb0			DDR2 667：0x5f7f1515
WR_DQS_SHIFT	62:56	0x0	0x0-0x7f	定义写数据时 clk_wr 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 CLK_WR 的目的是使发出的数据时钟提前四分之一个周期，从而比 DQS 提前四分之一个周期。
DQS_OUT_SHIFT	54:48	0x0	0x0-0x7f	定义写数据时 DQS 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使发出的 DQS 与时钟对齐。从而可以落在数据的中心
DLL_DQS_DELAY_8	46:40	0x0	0x0-0x7f	定义读数据时 DQS8 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
DLL_DQS_DELAY_7	38:32	0x0	0x0-0x7f	定义读数据时 DQS7 的延迟百分比，每次增加一个时钟周期的 1/128，延迟 DQS 的目的是使延迟后的 DQS 可以落在数据信号的中心
CONF_CTL_12[31:0]	Offset: 0xc0			DDR2 667：0x15000000
TRAS_MIN	31:24	0x0	0x0-0xff	定义内存模组行地址有效命令的最小时钟周期数
OUT_OF_RANGE_LENGTH	23:16	0x0	0x0-0xff	发生越界访问时的命令长度（只读）
ECC_U_SYND	15:8	0x0	0x0-0xff	发生 2bit 不可纠错误时的原因（只读）
ECC_C_SYND	7:0	0x0	0x0-0xff	发生 1bit 可纠错错误时的原因（只读）
CONF_CTL_12[63:32]	Offset: 0xc0			DDR2 667：0x002a3c05
DLL_DQS_DELAY_BYPASS_0	56:48	0x0	0x0-0x1ff	定义 DLL bypass 模式下 dqs0 延迟线的个数

TRFC	47:40	0x0	0x0-0xff	定义内存模组刷新操作需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。
TRCD_INT	39:32	0x0	0x0-0xff	定义内存模组 RAS 到 CAS 之间的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。
CONF_CTL_13[31:0]	Offset: 0xd0			DDR2 667：0x002a002a
DLL_DQS_DELAY_BYPASS_2	24:16	0x0	0x0-0x1	定义 DLL bypass 模式下 dqs2 延迟线的个数
DLL_DQS_DELAY_BYPASS_1	8:0	0x0	0x0-0x1	定义 DLL bypass 模式下 dqs1 延迟线的个数
CONF_CTL_13[63:32]	Offset: 0xd0			DDR2 667：0x002a002a
DLL_DQS_DELAY_BYPASS_4	56:48	0x0	0x0-0x1ff	定义 DLL bypass 模式下 dqs4 延迟线的个数
DLL_DQS_DELAY_BYPASS_3	40:32	0x0	0x0-0x1ff	定义 DLL bypass 模式下 dqs3 延迟线的个数
CONF_CTL_14[31:0]	Offset: 0xe0			DDR2 667：0x002a002a
DLL_DQS_DELAY_BYPASS_6	24:16	0x0	0x0-0x1ff	定义 DLL bypass 模式下 dqs6 延迟线的个数
DLL_DQS_DELAY_BYPASS_5	8:0	0x0	0x0-0x1ff	定义 DLL bypass 模式下 dqs5 延迟线的个数
CONF_CTL_14[63:32]	Offset: 0xe0			DDR2 667：0x002a002a
DLL_DQS_DELAY_BYPASS_8	56:48	0x0	0x0-0x1ff	定义 DLL bypass 模式下 dqs8 延迟线的个数
DLL_DQS_DELAY_BYPASS_7	40:32	0x0	0x0-0x1ff	定义 DLL bypass 模式下 dqs7 延迟线的个数
CONF_CTL_15[31:0]	Offset: 0xf0			DDR2 667：0x00000004
DLL_LOCK	24:16	0x0	0x0-0x1ff	指示 DLL 锁定时，延迟整个时钟周期所使用的延迟单元个数（只读）
DLL_INCREMENT	8:0	0x0	0x0-0x1ff	定义 DLL 进行锁定时，每次调整增加的延迟单元个数，不应为零，设小精度会比较高
CONF_CTL_15[63:32]	Offset: 0xf0			DDR2 667：0x00b4000a
DQS_OUT_SHIFT_BYPASS	56:48	0x0	0x0-0x1ff	定义 dqs out bypass 模式下 wr_dqs 延迟单元数
DLL_START_POINT	40:32	0x0	0x0-0x1ff	定义 DLL 进行锁定时，延迟单元起始个数， 不应小于 4，不应过大，否则会使访存出错。
CONF_CTL_16[31:0]	Offset: 0x100			DDR2 667：0x00000087
INT_ACK	25:16	0x0	0x0-0x3ff	将参数中位设置为 1 时，将会使对应位的清中断
WR_DQS_SHIFT_BYPASS	8:0	0x0	0x0-0x1ff	定义 wr dqs bypass 模式下 wr_clk 延迟单元数
CONF_CTL_16[63:32]	Offset: 0x100			DDR2 667：0x00000000
INT_STATUS	58:48	0x0	0x0-0x7ff	内存控制器发生中断的原因（只读） 位：0 – 一条访存指令地址超出内存实际物理空间 位：1 – 多条访存指令地址超出内存实际物理空间 位：2 – 一次 ECC 一位错发生

				位：3 – 多次 ECC 一位错发生 位：4 – 一次 ECC 两位错发生 位：5 – 多次 ECC 两位错发生 位：6 – 控制器地址通道发生错误 位：7 – 控制器数据通道发生错误 位：8 – 内存初始化完成 位：9 – DLL 未锁定 位：10 – 以上任何一种中断发生
INT_MASK	42:32	0x0	0x0-0x7ff	内存控制器中断的掩码位，与 INT_STATUS 位一一对应
CONF_CTL_17[31:0]	Offset: 0x110			DDR2 667：0x0000181b
EMRS1_DATA	30:16	0x0	0x0-0x7ff	定义内存控制器初始化内存模组时写入到内存模组 EMRS1 寄存器中的数据
TREF	13:0	0x0	0x0-0x3ff	定义内存模组两次刷新命令的时钟间隔，需要根据具体内存颗粒及运行频率进行配置。
CONF_CTL_17[63:32]	Offset: 0x110			DDR2 667：0x00000000
EMRS2_DATA_1	62:48	0x0000	0x0-0x7fff	定义内存模组初始化时，片选 1 对应的 EMRS2 数据
EMRS2_DATA_0	46:32	0x0000	0x0-0x7fff	定义内存模组初始化时，片选 0 对应的 EMRS2 数据
CONF_CTL_18[31:0]	Offset: 0x120			DDR2 667：0x00000000
EMRS2_DATA_3	30:16	0x0000	0x0-0x7fff	定义内存模组初始化时，片选 3 对应的 EMRS2 数据
EMRS2_DATA_2	14:0	0x0000	0x0-0x7fff	定义内存模组初始化时，片选 2 对应的 EMRS2 数据
CONF_CTL_18[63:32]	Offset: 0x120			DDR2 667：0x001c0000
AXI0_EN_LT_WIDTH_INSTR	63:48	0x0000	0x0-0xffff	定义 AXI0 端口是否接收小于 64 位位宽的内存访问
EMRS3_DATA	46:32	0x0000	0x0-0x7fff	定义内存模组初始化时 EMRS3 对应的数据
CONF_CTL_19[31:0]	Offset: 0x130			DDR2 667：0x00c8006b
TDLL	31:16	0x0000	0x0-0xffff	定义内存模组 DLL 锁定需要的时钟周期数
TCPD	15:0	0x0000	0x0-0xffff	定义内存模组时钟有效到 precharge 之间的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。
CONF_CTL_19[63:32]	Offset: 0x130			DDR2 667：0x48e10002
TRAS_MAX	63:48	0x0000	0x0-0xffff	定义内存模组行有效命令的最大时钟周期数，需要根据具体内存颗粒及运行频率进行配置。
TPDEX	47:32	0x0000	0x0-0xffff	定义内存模组掉电退出命令的时钟周期数
CONF_CTL_20[31:0]	Offset: 0x140			DDR2 667：0x00c8002f

TXSR	31:16	0x0000	0x0-0xffff	定义内存模组自刷新退出需要的时钟周期数
TXSNR	15:0	0x0000	0x0-0xffff	定义内存模组 tXSNR 参数
CONF_CTL_20[63:32]	Offset: 0x140			DDR2 667：0x00000000
XOR_CHECK_BITS	63:48	0x0000	0x0-0xffff	当 fwc 参数设定时，下次写操作的 check bit 将会与该参数进行异或后写入内存
VERSION	47:32	0x2041	0x2041	定义内存控制器版本号（只读）
CONF_CTL_21[31:0]	Offset: 0x150			DDR2 667：0x00030d40
ECC_C_ADDR[7:0]	31:24	0x0000	0x0-0x1fffffff	记录发生 1bit ECC 错误时的地址信息（只读）
TINIT	23:0	0x0000	0x0-0xffff	定义内存模组初始化需要的时钟周期数，需要根据具体内存颗粒及运行频率进行配置。一般为 200us
CONF_CTL_21[63:32]	Offset: 0x150			DDR2 667：0x00000000
ECC_C_ADDR[36:8]	60:32	0x0	0x0-0x1fffffff	记录发生 1bit ECC 错误时的地址信息（只读）
CONF_CTL_22[31:0]	Offset: 0x160			DDR2 667：0x00000000
ECC_U_ADDR[31:0]	31:0	0x0	0x0-0x1fffffff	记录发生 2bit ECC 错误时的地址信息（只读）
CONF_CTL_22[63:32]	Offset: 0x160			DDR2 667：0x00000000
ECC_U_ADDR[36:32]	36:32	0x0	0x0-0x1fffffff	记录发生 2bit ECC 错误时的地址信息（只读）
CONF_CTL_23[31:0]	Offset: 0x170			DDR2 667：0x00000000
OUT_OF_RANGE_ADDR[31:0]	31:0	0x0	0x0-0x1fffffff	记录发生越界访问时的地址信息（只读）
CONF_CTL_23[63:32]	Offset: 0x170			DDR2 667：0x00000000
OUT_OF_RANGE_ADDR[36:32]	36:32	0x0	0x0-0x1fffffff	记录发生越界访问时的地址信息（只读）
CONF_CTL_24[31:0]	Offset: 0x180			DDR2 667：0x00000000
PORT_CMD_ERROR_ADDR[31:0]	31:0	0x0	0x0-0x1fffffff	记录端口发生命令错误时的地址信息（只读）
CONF_CTL_24[63:32]	Offset: 0x180			DDR2 667：0x00000000
PORT_CMD_ERROR_ADDR[36:32]	36:32	0x0	0x0-0x1fffffff	记录端口发生命令错误时的地址信息（只读）
CONF_CTL_25[31:0]	Offset: 0x190			DDR2 667：0x00000000
ECC_C_DATA[31:0]	31:0	0x0	0x0-0x1ffff	记录发生 1bit ECC 错误时的数据信息（只读）

			fff	
CONF_CTL_25[63:32] Offset: 0x190				DDR2 667：0x00000000
ECC_C_DATA[63:32]	63:32	0x0	0x0-0x1fffff fff	记录发生 1bit ECC 错误时的数据信息（只读）
CONF_CTL_26[31:0] Offset: 0x1a0				DDR2 667：0x00000000
ECC_U_DATA[31:0]	31:0	0x0	0x0-0x1fffff fff	记录发生 2bit ECC 错误时的数据信息（只读）
CONF_CTL_26[63:32] Offset: 0x1a0				DDR2 667：0x00000000
ECC_U_DATA[63:32]	63:32	0x0	0x0-0x1fffff fff	记录发生 2bit ECC 错误时的数据信息（只读）
CONF_CTL_27[31:0] Offset: 0x1b0				DDR2 667：0x00000000
CKE_DELAY	2:0	0x0	0x0-0x7	CKE 有效延迟
CONF_CTL_28[31:0] Offset: 0x1c0				DDR2 667：0x00000001
UB_DIMM	0:0	0x0	0x0-0x1	是否采用 unbuffered 内存条，对于普通内存条，应当置为 1，对于笔记本内存或是直接使用内存颗粒，应当置为 0，否则不能正常工作

对上述配置寄存器中一些位的说明如下：

(1) CONF_CTL_00 AP

此参数用于控制是否启用 Autoprecharge 功能，一旦启用 Autoprecharge，内存将在每条读写指令后关闭所操作页。在发送大量连续地址操作的时候如果打开此参数会导致性能的下降。

(2) CONF_CTL_00 CONCURRENTAP

此参数用于设置内存是否支持 Concurrent Autoprecharge，需要注意的是很多厂商的内存颗粒并不支持这种方式。

(3) CONF_CTL_03 SREFRESH

此参数用于设置内存进入 Self Refresh 工作方式。需要从 Self Refresh 方式返回时必须使这个参数置 0。

(4) CONF_CTL_07 CASLAT_LIN_GATE

此参数用于控制读数据返回时内存控制器对数据采样的时机，一般等于 CASLAT_LIN。根据主板走线带来的时钟信号和 DQS 信号间的偏差可加减 1。

(5) CONF_CTL_15 DLL_INCREMENT

此参数不应设为 0。

(6) CONF_CTL_15 DLL_START_POINT

此参数不应设为 0 或 1。且应小于锁定时得到正确 DLL_LOCK_VALUE 的 1.5

倍。

(7) CONF_CTL_28　　UB_DIMM

此参数在使用 Unbuffered 内存条时，需要设置为 1，在直接使用内存颗粒时需要设置为 0。

11 集成 IO 控制器

11.1 IO 控制器功能概述

龙芯 2F 在片内集成了 PCIX 控制器、Local IO 控制器、GPIO、中断控制器和部分显示加速功能。这些控制器共享交叉开关的一个从端口，见图 11-1。来自 CPU 核的请求经过交叉开关后，根据地址发送到相应控制器，地址空间如表 11-1 所示。

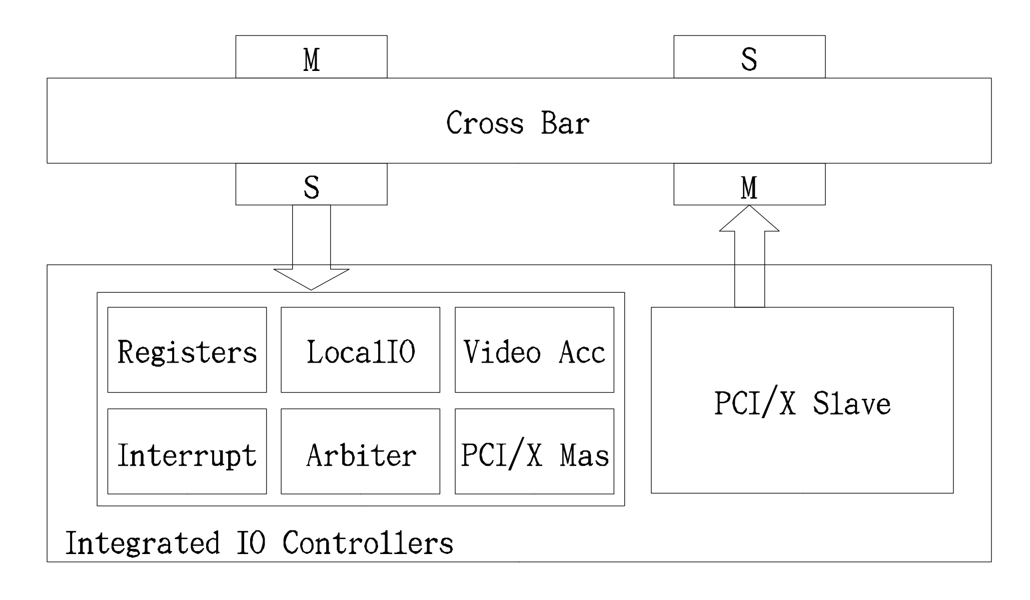


图 11-1 IO 控制器结构

表 11-1 IO 地址空间分配

开始地址	大小	所属空间	允许的访问类型	备注
0x00000000	256M	-	-	
0x10000000	64M	PCI MEM Lo0	CDWHB[1]	2
0x13f00000	1M	Video Acc	CDWHB	4
0x14000000	64M	PCI MEM Lo1	CDWHB	2



0x18000000	64M	PCI MEM Lo2	CDWHB	2
0x1c000000	32M	LIO ROM	CDWHB	
0x1e000000	28M	LIO IO	CDWHB	
0x1fc00000	1M	BOOT ROM	CDWHB	
0x1fd00000	1M	PCI IO	WHB	
0x1fe00000	256B	Registers	WHB	
0x1fe00100	256B	PCI Header	WHB	
0x1fe80000	2K	PCI CONF	WHB	
0x1ff00000	1M	LIO IO	CDWHB	3
0x20000000	1023.5G	PCI MEM Hi	CDWHB	2

注：

1. C(Cache 块), D(双字), W(字), H(半字), B(字节)。
2. 是否接受块读由 Mem_Win_Base, Mem_Win_Mask 寄存器控制。
3. 是否接受块读由 LIOCfg 寄存器控制。
4. 只写空间，当视频加速模块没有使能时不存在。

11.1.1 **PCIX 控制器**

龙芯 2F 的 PCIX 控制器既可以作为主桥控制整个系统，也可以作为普通 PCIX 设备工作在 PCIX 总线上。它的实现符合 PCI-X 1.0b 和 PCI 2.3 规范，配置头位于 0x1fe00000 开始的 256 字节，如表 11-2 所示。

表 11-2 PCIX 控制器配置头

字节 3	字节 2	字节 1	字节 0	地址
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	CacheLine Size	0C
Base Address Register 0				10
Base Address Register 1				14
Base Address Register 2				18
Base Address Register 3				1C
Base Address Register 4				20
Base Address Register 5				24
				28
Subsystem ID		Subsystem Vendor ID		2C
				30

			Capabilities Pointer	34
				38
Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line	3C
Implementation Specific Register(ISR40)				40
Implementation Specific Register(ISR44)				44
Implementation Specific Register(ISR48)				48
Implementation Specific Register(ISR4C)				4C
Implementation Specific Register(ISR50)				50
Implementation Specific Register(ISR54)				54
Implementation Specific Register(ISR58)				58
				...
PCIX Command Register				E0
PCIX Status Register				E4

龙芯 2F 的 PCIX 控制器支持三个 64 位窗口，由{BAR1, BAR0}、{BAR3, BAR2}、{BAR5, BAR4}三对寄存器配置窗口 0、1、2 的基址。窗口的大小、使能以及其它细节由另外三个对应寄存器 PCI_Hit0_Sel, PCI_Hit1_Sel, PCI_Hit2_Sel 控制，具体位域请参见表 11-6。

表 11-3 中断控制寄存器

位域	字段名	访问	复位值	说明
ISR_40				
31	tar_read_io	读写 (写 1 清)	0	target 端收到对 IO 或者是不可预取区域的访问
30	tar_read_discard	读写 (写 1 清)	0	target 端的 delay 请求被丢弃
29	tar_resp_delay	读写	0	target 访问何时给出 delay/split 0: 超时后 1: 马上
28	tar_delay_retry	读写	0	target 访问重试策略 0: 根据内部逻辑（见 29 位） 1: 马上重试
27	tar_read_abort_en	读写	0	若 target 对内部的读请求超时，是否让以 target-abort 回应

26:25	Reserved	-	0	
24	tar_write_abort_en	读写	0	若 target 对内部的写请求超时，是否让以 target-abort 回应
23	tar_master_abort	读写	0	是否允许 master-abort
22:20	tar_subseq_timeout	读写	000	target 后续延迟超时 000: 8 周期 其它：不支持
19:16	tar_init_timeout	读写	0000	target 初始延迟超时 PCI 模式下 0: 16 周期 1-7: 禁用计数器 8-15: 8-15 周期 PCIX 模式下超时计数固定为 8 周期，此处配置影响最大的 delay 访问数 0: 8 delay 访问 8: 1 delay 访问 9: 2 delay 访问 10: 3 delay 访问 11: 4 delay 访问 12: 5 delay 访问 13: 6 delay 访问 14: 7 delay 访问 15: 8 delay 访问
15:4	tar_pref_boundary	读写	000h	可预取边界配置（以 16 字节为单位） FFF: 64KB 到 16byte FFE: 64KB 到 32byte FF8: 64KB 到 128byte
3	tar_pref_bound_en	读写	0	使用 tar_pref_boundary 的配置 0: 预取到设备边界 1: 使用 tar_pref_boundary
2	Reserved	-	0	
1	tar_splitw_ctrl	读写	0	target split 写控制 0: 阻挡除 Posted Memory Write 以外的访问

				1：阻挡所有访问，直至 split 完成
0	mas_lat_timeout	读写	0	禁用 mater 访问超时 0：允许 master 访问超时 1：不允许
ISR_44				
31:0	Reserved	-	-	
ISR_48				
31:0	tar_pending_seq	读写	0	target 未处理完的请求号位向量 对应位写 1 可清
ISR_4C				
31:30	Reserved	-	-	
29	mas_write_defer	读写	0	允许后续的读越过前面未完成的写 (只对 PCI 有效)
28	mas_read_defer	读写	0	允许后续的读写越过前面未完成的读 (只对 PCI 有效)
27	mas_io_defer_cnt	读写	0	在外的最大 IO 请求数 0：由控制 1: 1
26:24	mas_read_defer_cnt	读写	010	master 支持在外读的最大数(只对 PCI 有效） 0: 8 1-7: 1-7 注：一个双地址周期访问占两项
23:16	err_seq_id	只读	00h	target/master 错误号
15	err_type	只读	0	target/master 出错的命令类型 0:
14	err_module	只读	0	出错的模块 0: target 1: master
13	system_error	读写	0	target/master 系统错（写 1 清）
12	data_parity_error	读写	0	target/master 数据奇偶错（写 1 清）
11	ctrl_parity_error	读写	0	target/master 地址奇偶错（写 1 清）
10:0	Reserved	-	-	

ISR_50				
31:0	mas_pending_seq	读写	0	master 未处理完的请求号位向量 对应位写 1 可清
ISR_54				
31:0	mas_split_err	读写	0	split 返回出错的请求号位向量
ISR_58				
31:30	Reserved	-	-	
29:28	tar_split_priority	读写	0	target split 返回优先级 0 最高, 3 最低
27:26	mas_req_priority	读写	0	master 对外的优先级 0 最高, 3 最低
25	Priority_en	读写	0	仲裁算法（在 master 的访问和 target 的 split 返回间做仲裁） 0: 固定优先级 1: 轮转
24:18	保留	-	-	
17	mas_retry_aborted	读写	0	master 重试取消（写 1 清）
16	mas_trdy_timeout	读写	0	master TRDY 超时计数
15:8	mas_retry_value	读写	00h	master 重试次数 0: 无限重试 1-255: 1-255 次
7:0	mas_trdy_count	读写	00h	master TRDY 超时计数器 0: 禁用 1-255: 1-255 拍

在发起配置空间读写前，应用程序应先配置好 PCIMap_Cfg 寄存器，告诉控制器欲发起的配置操作的类型和高 16 位地址线上的值。然后对 0x1fe80000 开始的 2K 空间进行读写即可访问对应设备的配置头。设备号由根据 PCIMap_Cfg[15:0]从低到高优先编码得到。

配置操作地址生成见图 11-2。

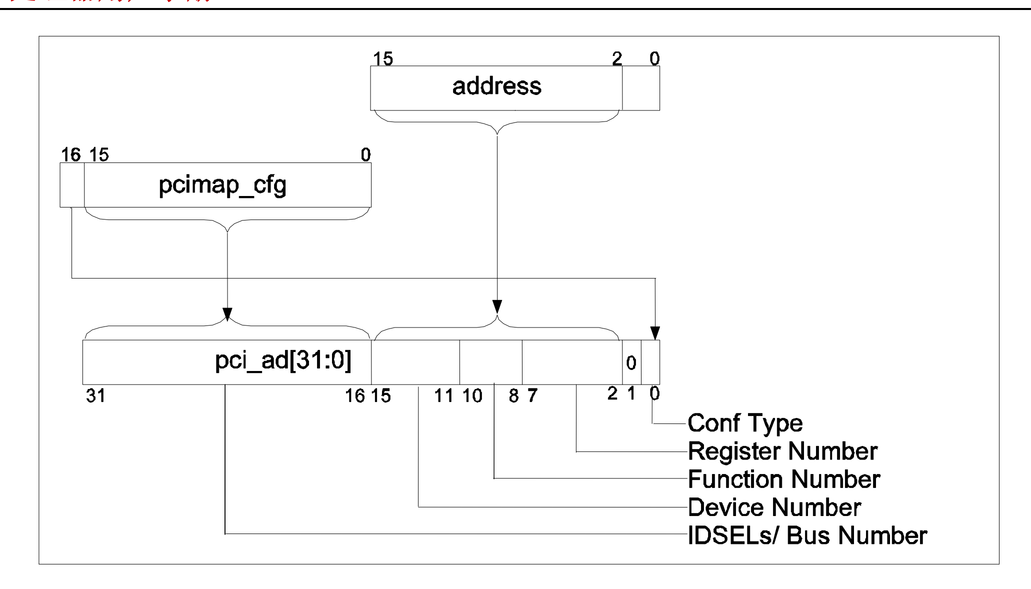


图 11-2 配置读写总线地址生成

11.1.2 LocalIO 控制器

LocalIO 控制器提供了简单外设访问接口，主要用于连接系统启动 ROM。它对外提供两个片选，具有可配置的数据位宽和访问延迟（配置寄存器见 CR08 LIOCfg）。其中 wait 参数指 liord 或 liowr 信号为低的周期数减 1，读写时序可参考图 11-3，图 11-4。当数据位宽为 16 时，送出的地址由 CPU 物理地址右移一位得到。

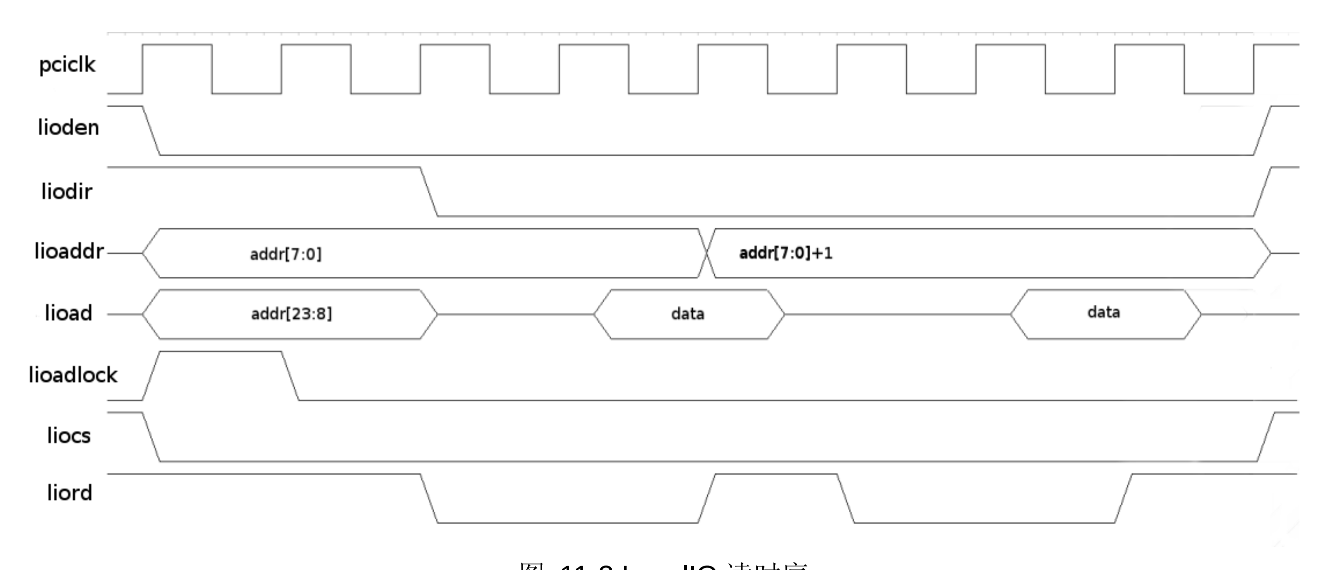


图 11-3 LocalIO 读时序

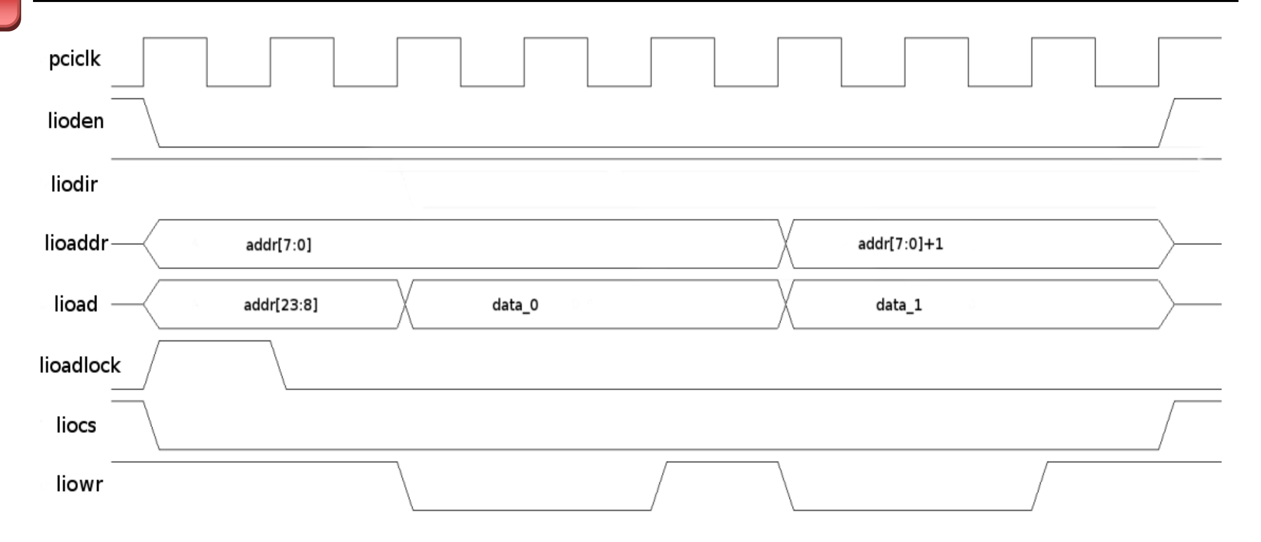


图 11-4 LocalIO 写时序

11.1.3 中断控制器

来自 CPU 核外的中断由集成中断控制器进行管理。每根中断线都有以下属性：使能、有效电平和触发方式。所有外部中断的有效电平均可设置，并复位为低有效。CP0 寄存器定义了 6 个外部位，其中低 4 位直接来自 INT3-0，它们的使能在中断控制器中为 1 且不可更改；其它中断源通过第 5 根中断线送到 CPU 核；第 6 根中断线恒为 0，用于内部时钟中断。

中断相关配置寄存器都是以位的形式对相应的中断线进行控制，中断控制位连接及属性配置见表 11-3。中断的有效电平由 Intpol 寄存器设置。中断使能（Enable）的配置有三个寄存器：Intenset、Intenclr 和 Inten。Intenset 设置中断使能，Intenset 寄存器写 1 的位对应的中断被使能。Intenclr 清除中断使能，Intenclr 寄存器写 1 的位对应的中断被清除。Inten 寄存器读取当前各中断使能的情况。脉冲形式的中断信号（如 PCI_SERR）由 Intedge 配置寄存器来选择，写 1 表示脉冲触发，写 0 表示电平触发。中断处理程序可以通过 Intenclr 的相应位来清除脉冲记录。

表 11-4 中断控制寄存器

位域	控制寄存器（访问属性/缺省值）					中断源
	Intpol	Intedge	Inten	Intenset	Intenclr	
3 : 0	RW / 0	RW / 0	RO / 0	WO / 0	WO / 0	GPIO
7 : 4	RO / 0	RO / 0	RO / 0	WO / 0	WO / 0	PCI_INTn
8	RO / 1	RO / 1	RO / 0	WO / 0	WO / 0	PCI_PERR
9	RO / 1	RO / 1	RO / 0	WO / 0	WO / 0	PCI_SERR
10	RO / 1	RO / 0	RO / 0	WO / 0	WO / 0	DDR2 Controller
14 : 11	RW / 0	RW / 0	RO / 0	保留	WO / 0	INTn

31 : 15 保留

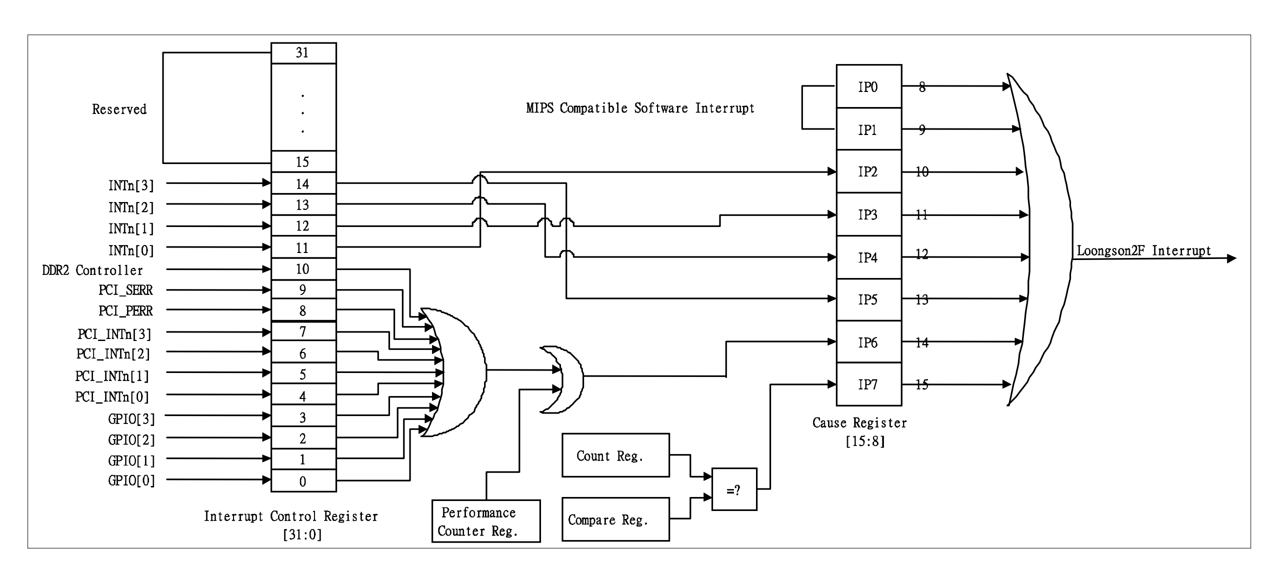


图 11-5 龙芯 2F 处理器中断路由示意图

11.1.4 PCI/PCIX 仲裁器

PCI/PCIX 仲裁器实现了两级轮转仲裁、总线停靠和损坏主设备的隔离。其配置和状态见 PXArb_Config 和 PXArb_Status 寄存器。PCI/PCIX 总线请求与应答线分配见表 11-4。

表 11-5 PCI/PCIX 总线请求与应答线分配

请求与应答线	描述
0	内部集成 PCI/PCIX 控制器
7:1	外部请求 6~0

基于轮转的仲裁算法提供两个级别，第二级整体作为第一级中的一员一起调度。当多个设备同时申请总线时每轮转完一次第一级设备，第二级中优先级最高的设备可以得到总线。

仲裁器被设计成任何时候只要条件允许就可以切换，对于一些不符合协议的 PCI 设备，这样做可能会使之不正常。使用强制优先级可以让这些设备通过持续请求来占有总线。

总线停靠是指当没有设备请求使用总线时是否选择其中一个给出允许信号。对于已经得到允许的设备而言，直接发起总线操作能够提高效率。龙芯 2F 的 PCI 仲裁器提供两种停靠模式：最后一个主设备和默认主设备。如果在特殊场合下不能够停靠，可以将仲裁器设置为停靠到默认 0 号主设备（内部控制器），且依靠延迟为 0。

11.1.5 显示加速

视频播放中色彩空间转换和缩放运算非常简单，但是会消耗通用 CPU 大量计算资源。

因此，龙芯 2F 在片上集成了部分显示加速的功能。这个功能在没有视频加速功能的显卡上（例如 XGI 的 Z9 显示控制芯片），能节省大量的 CPU 运算资源。这部分模块加在 CPU 写往 PCIX 控制器的数据通路中。在该功能使能时，写往 0x13f00000~0x13ffffff 的数据会当作视频原始帧数据进行色彩空间转换和缩放等处理。数据通路见图 11-5。

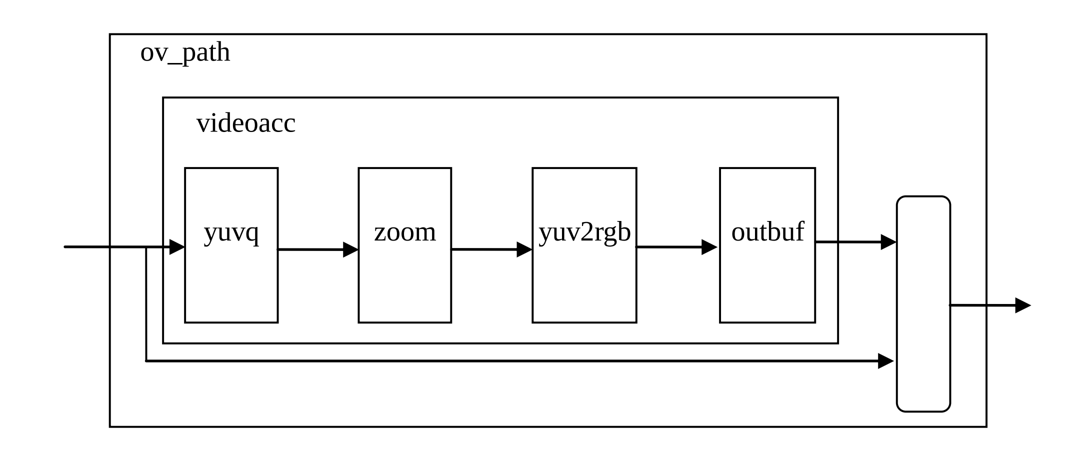


图 11-6 显示加速模块数据通路

软件设计人员在使用显示加速时需要足够小心，不恰当的配置及使用将会导致系统灾难性的后果。GenCfg 寄存器中第 0 位的 ov_en 是全局使能位，当它为 0 时显示加速模块不可用。当正在使用显示加速功能时，操作系统必须保证这一位不能清零。

YUV422 模式下，显示数据以 64 个点为单位写入。第一个 32 字节为前 32 点的 Y 的数据；第二个 32 字节为 64 点的 U 数据；第三个 32 字节为 64 点的 V 数据；第四个 32 字节为后 32 点的 Y 的数据。

YUV444 模式下，显示数据同样以 64 个点为单位写入。第一个 32 字节为前 32 点的 Y 的数据；第二个 32 字节为 32 点的 U 数据；第三个 32 字节为 32 点的 V 数据。然后再按同样次序写后 32 个点的 YUV 共 96 字节。

更改相关寄存器配置前，必须把 ov_ctrl.reset 置位，设置完成后清零。

为得到更加明显的加速效果，播放程序需要使用 Uncached Accelerated 的 TLB 映射。具体设置参见参考代码一章。

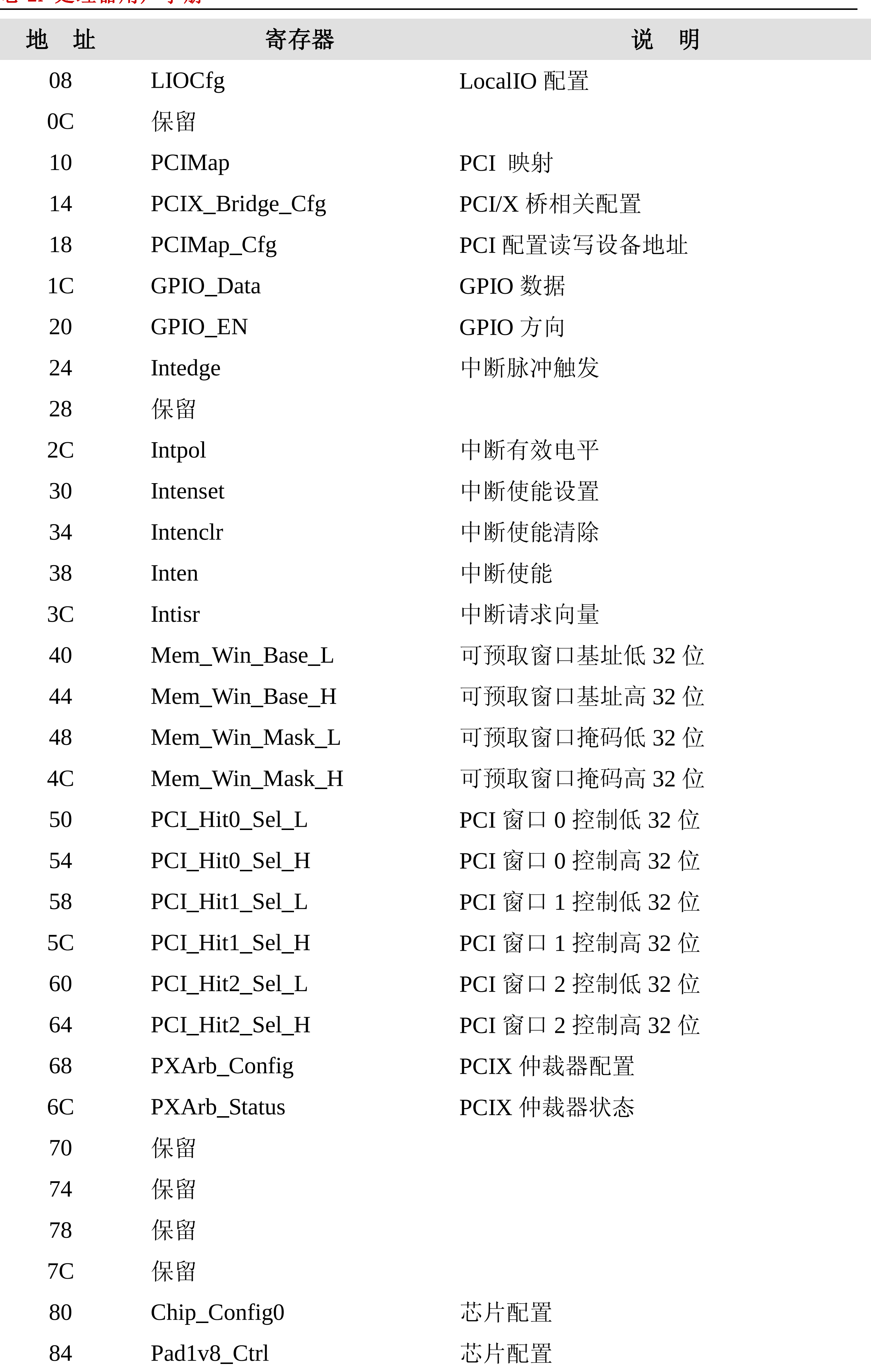
11.2 寄存器描述

11.2.1 IO 控制寄存器

除了 PCI 头外所有的可配置部分都位于 0x1fe00100 的寄存器中。表 11-5 列出了所有这些寄存器，表 11-6 给出寄存器的详细说明。

表 11-6 IO 控制寄存器

地　址	寄存器	说　明
00	PonCfg	上电配置
04	GenCfg	常规配置



地　址	寄存器	说　明
08	LIOCfg	LocalIO 配置
0C	保留	
10	PCIMap	PCI 映射
14	PCIX_Bridge_Cfg	PCI/X 桥相关配置
18	PCIMap_Cfg	PCI 配置读写设备地址
1C	GPIO_Data	GPIO 数据
20	GPIO_EN	GPIO 方向
24	Intedge	中断脉冲触发
28	保留	
2C	Intpol	中断有效电平
30	Intenset	中断使能设置
34	Intenclr	中断使能清除
38	Inten	中断使能
3C	Intisr	中断请求向量
40	Mem_Win_Base_L	可预取窗口基址低 32 位
44	Mem_Win_Base_H	可预取窗口基址高 32 位
48	Mem_Win_Mask_L	可预取窗口掩码低 32 位
4C	Mem_Win_Mask_H	可预取窗口掩码高 32 位
50	PCI_Hit0_Sel_L	PCI 窗口 0 控制低 32 位
54	PCI_Hit0_Sel_H	PCI 窗口 0 控制高 32 位
58	PCI_Hit1_Sel_L	PCI 窗口 1 控制低 32 位
5C	PCI_Hit1_Sel_H	PCI 窗口 1 控制高 32 位
60	PCI_Hit2_Sel_L	PCI 窗口 2 控制低 32 位
64	PCI_Hit2_Sel_H	PCI 窗口 2 控制高 32 位
68	PXArb_Config	PCIX 仲裁器配置
6C	PXArb_Status	PCIX 仲裁器状态
70	保留	
74	保留	
78	保留	
7C	保留	
80	Chip_Config0	芯片配置
84	Pad1v8_Ctrl	芯片配置



地　址	寄存器	说　明
88	Pad3v3_Ctrl	芯片配置
8C	保留	
90	Comp_Code	芯片采样
94	Chip_Sample1	芯片采样
98	保留	
9C	保留	
A0	OV_Ctrl	显示加速控制
A4	OV_Ori_Size	原始图象尺寸
A8	OV_Zoom_Size	缩放后图象尺寸
AC	OV_Fb_Base	当前显示图象 Frame Buffer 起始地址
B0	OV_Fb_Stride	当前显示图象 Frame Buffer 横向跨度
B4	OV_Hor_Zoom1	水平缩放控制 1
B8	OV_Hor_Zoom2	水平缩放控制 2
BC	OV_Ver_Zoom	垂直缩放控制
C0	OV_X_Pos	显示图象的 X 点坐标
C4	OV_X_Width	屏幕的横向宽度
C8	OV_Fb_Base	Frame Buffer 起始地址
CC	OV_Fb_Mask	Frame Buffer 范围掩码

表 11-7 寄存器详细描述

位域	字段名	访问	复位值	说明
CR00: PonCfg				
15:0	pcix_bus_dev	只读	lio_ad[7:0]	PCIX Agent 模式下 CPU 取指所使用的总线、设备号
15:8	保留	只读	lio_ad[15:8]	
23:16	pon_pci_configi	只读	pci_configi	PCI_Configi 引脚值
31:24	保留	只读		
CR04: GenCfg				
0	ov_en	读写	0	视频加速使能
31:1	保留	只读	0	
CR08: LIOCfg				
1:0	保留	只读	0	

位	名称	读写	缺省值	描述
6:2	rom_wait	读写	5'b11111	Rom 数据读取延迟（PCI 时钟周期数）
7	rom_width	读写	pci_config[0]	Rom 数据位宽 0：8 位 1：16 位
12:8	io_wait	读写	5'b11111	IO 数据读取延迟（PCI 时钟周期数）
13	io_width	读写	1'b0	IO 数据位宽 0：8 位 1：16 位
14	iopf_en	读写	1'b0	IO 设备可预取使能 0：禁止 Cache 块读 1：允许 Cache 块读
31:15	保留	只读	0	
CR10: PCIMap				
5:0	trans_lo0	读写	0	PCI_Mem_Lo0 窗口映射地址高 6 位
11:6	trans_lo1	读写	0	PCI_Mem_Lo1 窗口映射地址高 6 位
17:12	trans_lo2	读写	0	PCI_Mem_Lo2 窗口映射地址高 6 位
31:18	保留	只读	0	
CR14: PCIX_Bridge_Cfg				
5:0	pcix_rgate	读写	6'h18	PCIX 模式下向 DDR2 发读取数门限
6	pcix_ro_en	读写	0	PCIX 桥是否允许写越过读
31:18	保留	只读	0	
CR18: PCIMap_Cfg				
15:0	dev_addr	读写	0	PCI 配置读写时 AD 线高 16 位
16	conf_type	读写	0	配置读写的类型
31:17	保留	只读	0	
CR1C: GPIO_Data				
3:0	gpio_out	读写	0	GPIO 输出数据
15:4	保留	只读	0	
19:16	gpio_in	读写	0	GPIO 输入数据
31:20	保留	只读	0	
CR20: GPIO_EN				
3:0	gpio_en	读写	F	高为输入，低输出

位	名称	访问	复位值	说明
31:4	保留	只读	0	
CR3C: Intisr				
14:0	中断线状态	只读	0	1 表示有中断，0 表示没有中断
31:15	保留	只读	0	
CR24,2C,30,34,38: Intedge,Intpol,Intenset,Intenclr,Inten				
见表 11-3				
CR50,54/58,5C/60,64: PCI_Hit*_Sel_*				
0	保留	只读	0	
2:1	pci_img_size	读写	2'b11	00: 32 位；10: 64 位；其它: 无效
3	pref_en	读写	0	预取使能
11:4	保留	只读	0	
62:12	bar_mask	读写	0	窗口大小掩码（高位 1，低位 0）
63	burst_cap	读写	1	是否允许突发传送
CR68:PXArb_Config				
0	device_en	读写	1	外部设备允许
1	disable_broken	读写	0	禁用损坏的主设备
2	default_mas_en	读写	1	总线停靠到默认主设备 0：停靠到最后一个主设备 1：停靠到默认主设备
5:3	default_master	读写	0	总线停靠默认主设备号
7:6	park_delay	读写	2'b11	从没有设备请求总线开始到触发停靠默认设备行为的延迟 00: 0 周期 01: 8 周期 10: 32 周期 11: 128 周期
15:8	level	读写	8'h01	处于第一级的设备
23:16	rude_dev	读写	0	强制优先级设备 为 1 的位对应的 PCI 设备在得到总线后可以通过持续请求来占住总线
31:13	保留	只读	0	
CR6C: PXArb_Status				
7:0	broken_master	只读	0	损坏的主设备（改变禁用策略时清零）

10:8	Last_master	只读	0	最后使用总线的主设备
31:11	保留	只读	0	
CR80: Chip_config0				
2:0	Freq_scale	读写	3'b111	与处理器核时钟频率对应关系： 7: 8/8 全速 6: 7/8 5: 6/8 4: 5/8 3: 4/8 2: 3/8 1: 2/8 0: 0/8 stand-by 进入 stand-by 状态后，外部中断可以将 freq_scale 恢复成 1。其中外部中断由 IP6-IP2 组成，高有效。
3	disable_scache	读写	0	禁用二级 Cache
4	imp_first	读写	1	关键字优先
7:5	保留	读写	0	
8	disable_ddr_conf	读写	0	禁用 DDR2 配置端口
9	ddr_buffer_cpu	读写	1	写结束时是否允许数据未到内存
10	ddr_buffer_pci	读写	1	写结束时是否允许数据未到内存
31:11	保留	只读	0	
CR84: Pad1v8_Ctrl				
0	compen	读写	0	DDR2 pad compensation cell 工作模式设置（见下表 11-7）
1	comptq	读写	0	DDR2 pad compensation cell 工作模式设置（见下表 11-7）
2	freeze	读写	0	DDR2 pad compensation cell 工作模式设置（见下表 11-7）
3	accurate	读写	0	DDR2 pad compensation cell 工作模式设置（见下表 11-7）
10:4	Nasrc	读写	7'b1111000	DDR2 pad compensation cell READ 模式下补偿编码输入（见下表 11-7）
11	Proga	读写	1	DDR2 pad 期望工作频率设置（见下表 11-8）
12	Progb	读写	0	DDR2 pad 期望工作频率设置（见下

				表 11-8）
13	Mod	读写	0	DDR2 pad 输入电平标准选择 (0:SSTL18 1:CMOS）
14	Strb	读写	0	DDR2 DQS pad 输出模式(0:差分 1:伪差分)
15	En	读写	0	DDR2 CK3-CK5 pad 输出使能(0:使能 1:不使能)
16	zoutproga	读写	0	DDR2 pad 期望工作频率设置（见下表 11-8）
CR88: Pad3v3_Ctrl				
0	compen	读写	0	3.3V pad compensation cell 工作模式设置（见下表 11-7）
1	comptq	读写	0	3.3V pad compensation cell 工作模式设置（见下表 11-7）
2	freeze	读写	0	3.3V pad compensation cell 工作模式设置（见下表 11-7）
3	accurate	读写	0	3.3V pad compensation cell 工作模式设置（见下表 11-7）
10:4	Nasrc	读写	7'b1111000	3.3V pad compensation cell READ 模式下补偿编码输入（见下表 11-7）
CR90: CompCode				
6:0	ddr2_asrc	只读		DDR2 pad compensation cell 补偿编码输出（见下表 11-7）
7	保留	只读		
14:8	pci_asrc	只读		PCI pad compensation cell 补偿编码输出（见下表 11-7）
15	保留	只读		
22:16	sys_asrc	只读		3.3V pad compensation cell 补偿编码输出（见下表 11-7）
32:23	保留	只读		
CR94: Chip_sample1				
9:0	sys_clksel	只读	sys_clksel	时钟倍频控制引脚值
31:10	保留	只读		

表 11-8 DDR2/PCI/3V3 pad Compensation cell 工作模式设置

compen	comptq	freeze	accurate	Compensation cell 工作模式	ddr2_asrc/pci_asrc/ sys_asrc 来源
0	0	0	x	normal	f(PVT)
1	1	x	x	read	Nasrc
0	0	1	x	freeze	锁定为前一个值
0	1	x	x	typical	0001111
1	0	x	x	HZ	三态
x	x	x	1	accurate	外置精密电阻补偿输入

表 11-9 DDR2 pad 期望工作频率设置

zoutproga	Proga	Progb	期望工作频率
1	x	1	166MHZ
0	0	0	200MHZ
0	0	1	266MHZ
0	1	0	333MHZ

11.2.2 显示加速控制寄存器

表 11-10 显示加速控制寄存器描述

位域	字段名	访问	复位值	说明
CRA0：OV_Ctrl（显示加速控制）				
0	reset	读写	0	复位显示加速模块 模块解复位后软件需要等足够长的时间（如 1 毫秒）才能写显示缓冲
1	Y2R_EN	读写	0	是否进行 YUV 到 RGB 的转换
2	ZoomEn	读写	0	是否进行硬件缩放
4:3	inFMT	读写	0	输入的图象格式 01: YUV422 10: YUV444
6:5	outFMT	读写	0	该模块送给显示控制器的图象格式 00: RGB16 01: RGB24 10: RGB32

10:7	resolution	读写	0	显示控制器的当前显示分辨率 0000: 320x200 0001: 320x350 0010: 360x400 0011: 640x200 0100: 640x350 0101: 640x480 0110: 720x350 0111: 720x400 1000: 800x600 1001:1024x768 1010:1280x1024 1011:1600x1200
CRA4：OV_Ori_Size（原始图象尺寸）				
10:0	X	读写	0	
21:11	Y	读写	0	
CRA8：OV_Zoom_Size（缩放后图象尺寸）				
10:0	X	读写	0	
21:11	Y	读写	0	
CRAC：OV_Fb_Base（当前显示图象的 Frame Buffer 的起始地址）				
31:0	addr	读写	0	当前显示图象的 Frame Buffer 的起始地址，软件上可以通过缩放比例以及驱动中的 dest_x，和 dest_y（当前窗口的开始位置），以及显卡 Frame Buffer 的起始地址等信息计算得出。
CRB0：OV_Fb_Stride（当前显示图象的 Frame Buffer 横向跨度）				
31:0	stride	读写	0	当前显示图象的 Frame Buffer 横向跨度
CRB4：OV_Hor_Zoom1（水平缩放控制 1）				
10:0	ov_seg_size	读写	0	每段包含点数。每段包含点数为：（32÷缩放比例+1）的向下取整整数。
27:11	ov_stepx	读写	0	缩放比例。缩放比例为水平方向的原始尺寸除以缩放后尺寸的比例，并且存放为 5.12 格式的一个小数。
CRB8：OV_Hor_Zoom2（水平缩放控制 2）				

10:0	ov_last_seg_size	读写	0	最后一段包含点数。从-0.5 开始计数，计算到结束点+0.5 的值除以 32 的余数。格式也为 5.12
28:11	ov_size_mul_step	读写	0	缩放比例×每段包含点数。前面两个寄存器的积
CRBC：OV_Ver_Zoom（垂直缩放控制）				
16:0	ov_stepy	读写	0	参见 ov_stepx 的定义。
CRC0：OV_X_Pos（显示图象的 x 点坐标）				
12:0	ov_x_pos	读写	0	显示图象的 x 点坐标，是一个有符号数。
CRC4：OV_X_Width（屏幕的横向宽度）				
10:0	ov_x_width	读写	0	屏幕的横向宽度
CRC8：OV_Fb_Base（Frame Buffer 起始地址）				
31:0	ov_fb_base	读写	0	Frame Buffer 起始地址
CRCC：OV_Fb_Mask（Frame Buffer 范围掩码）				
31:0	ov_fb_mask	读写	0	Frame Buffer 范围掩码 这个区域外的写将不送到 PCI 总线

11.2.3 PCI 桥地址映射

CPU 核可以从四个窗口发起 PCI 的 Memory 访问。其中，命中 PCI MEM Lo0/1/2 的请求在送到 PCI 总线时地址的高 6 位由相应的 trans_lo 替换；命中 PCI MEM Hi 窗口的请求在送到 PCI 总线时将不作转换。来自 PCI 总线的访问在送到 DDR2 控制器过程中由交叉开关进行地址变换。参见第 10 章。

配置举例：

下面以一个假想的使用两颗龙芯 2F 处理器组成的多处理器系统为例，介绍地址窗口的配置。这个系统由两块处理器子卡、北桥及其它 PCI 设备组成。每块处理器卡配置 4G 内存，均把其中 0xC0000000 开始的 1G 内存开放于 PCI 总线。PCI Memory 空间分配如下表

地址范围	大小	设备
0x00000000-0x03ffffff	64M	南桥
0x10000000-0x13ffffff	64M	网卡
0x1c000000-0x1fffffff	64M	北桥
0x40000000-0x7fffffff	1G	北桥
0x80000000-0xbfffffff	1G	处理器 0

0xc0000000-0xffffffff　1G　处理器 1

0 号处理器配置如下：

```
m0_win0_base = 0x00000000_00000000      // CPU 0-256M => DDR2 0~256M
m0_win0_mask = 0xffffffff_f0000000
m0_win0_mmap = 0x00000000_00000000

m0_win1_base = 0x00000000_10000000      // CPU 256~512M =>
m0_win1_mask = 0xffffffff_f0000000      // IO controller 256~512M
m0_win1_mmap = 0x00000000_10000001

m0_win2_base = 0x00000001_00000000      // CPU 4~8G => DDR2 0~4G
m0_win2_mask = 0xffffffff_00000000
m0_win2_mmap = 0x00000000_00000000

m0_win3_base = 0x00000002_00000000      // CPU 8G~12G =>
m0_win3_mask = 0xffffffff_00000000      // IO controller 0G~4G
m0_win3_mmap = 0x00000000_00000001      // => PCI 0G~4G

trans_lo0 = 0x00                        // lo0 => PCI 0~64M
trans_lo1 = 0x04                        // lo1 => PCI 256~320M
trans_lo2 = 0x07                        // lo2 => PCI 446~512M

pci_hit0_sel = 0xffffffff_c000000c      // 1G bar size
pci_hit1_sel = 0x00000000_00000006      // invalid bar
pci_hit2_sel = 0x00000000_00000006      // invalid bar
pci_bar0     = 0x00000000_80000000      // PCI 2G

m1_win0_base = 0x00000000_80000000      // PCI 2~3G => DDR2 3~4G
m1_win0_mask = 0xffffffff_c0000000
m1_win0_mmap = 0x00000000_c0000000
```

1 号处理器配置与 0 号基本一致，只是对 PCI 开的窗口位置不同：

```
pci_hit0_sel = 0xffffffff_c000000c          // 1G bar size
pci_hit1_sel = 0x00000000_00000006          // invalid bar
pci_hit2_sel = 0x00000000_00000006          // invalid bar
pci_bar0     = 0x00000000_c0000000          // PCI 3G

m1_win0_base = 0x00000000_c0000000          // PCI 3~4G => DDR2 3~4G
m1_win0_mask = 0xffffffff_c0000000
m1_win0_mmap = 0x00000000_c0000000
```

注：交叉开关的四个窗口不支持相互重叠。

12 性能优化

本章提供了龙芯 2F 体系结构中一些与软件性能优化相关的信息，包括指令延迟和指令循环的间隔、扩展指令、指令流和存储访问处理等，可供编译器和其它软件开发者参考。

12.1 用户指令的延迟和循环间隔

表 12-1 给出了在 ALU1/2, MEM, FALU1/2 功能单元中执行的所有用户指令的延迟和循环间隔，不包括内核指令和控制指令。这里的指令延迟是指从该指令发射到其结果能被下一条指令使用所需要的拍数（一个处理器周期为一拍）。例如，大部分的 ALU 指令延迟为 2,这表示 ALU 指令的结果要隔一拍后才能被后续指令使用。因此，形如 i = i + 1 的相关循环（下一个循环依赖上一个循环的结果）不能每拍出一个结果。而一个指令的循环间隔则是指功能部件接受这种指令的频度，1 表示每拍都能接受一个以上的同类指令，n 表示功能部件接受一个该指令后，需要等 n-1 拍后才能再接受同类指令。全流水功能部件的指令循环间隔为 1。

表 12-1 用户指令的延时和循环间隔

指令类型	执行部件	延迟	循环间隔
整 型 操 作			
ADD/SUB/Logical/Shift/LUI/CMP	ALU1/2	2	1
Trap/Branch	ALU1	2	1
MF/MT HI/LO	ALU1/2	2	1
(D)MULT(U)	ALU2	5	2[1]
(D)MULT(U)G	ALU2	5	1
(D)DIV(U)	ALU2	5-38[2]	10-76[3]
(D)DIV(U)G	ALU2	5-38	4-37
(D)MOD(U)G	ALU2	5-38	4-37
Load	MEM	5	1
Store	MEM	-	1
浮 点 操 作			
(D)MTC1/(D)MFC1	MEM	5	1
ABS/NEG/C.cond/BCLT/BC1F/Move/CVT*	FALU1	3	1

[1]内部被划分成两条指令

[2]延迟与操作数有关,可以用以下方法粗略估算：

(lz(a) < lz(b))?(lz(b)-lz(a))/2 + 4 – ez(c) / 2 : 1
其中 a/b=c, lz ： 前导 0 的个数；ez ： 后置 0 的个数

[3]内部被划分成两条指令，且不流水

Round/Trunc/Ceil/Floor/CVT*	FALU1	5	1
ADD/SUB/MUL/MADD/MSUB/NMADD/NMSUB	FALU1/2	7	1
DIV.S	FALU2	5-11	4-10
DIV.D	FALU2	5-18	4-17
SQRT.S	FALU2	5-17	4-16
SQRT.D	FALU2	5-32	4-31
LWC1/LDC1	MEM	5	1
SWC1/SDC1	MEM	-	1

对于表 12-1，还有以下几点说明：

● 这里的 Load/Store 操作的循环间隔并不包括 LL/SC 指令。LL/SC 是等待发射操作，只有当它们位于 Reorder 队列队首，而且此时 CP0 队列为空时，才可以被发射。

● 对于 HI/LO 寄存器，没有特别的使用限制。它们和其它的通用寄存器一样使用。这个表中并不包含 CTC1/CFC1。它们和许多其它的控制指令一样被序列化。

● 这个表中也不包含多媒体指令。因为它们是通过扩展普通浮点指令的格式而完成的，它们的功能单元和延时与被扩展的指令相同。

12.2 指令扩充

龙芯 2F 完成了以下几种指令扩充：

● 只写一个结果到通用寄存器的定点乘除。包括 12 条指令：

(D)MULTG, (D)MULTUG, (D)DIVG

(D)DIVUG, (D)MODG , (D)MODUG

在标准的 MIPS 指令集中，乘法和除法在一个操作中需要写两个特殊的结果寄存器(HI/LO)，它们在 RISC 流水线中很难实现。为了使用这些结果，将不得不使用额外的指令把它从 HI/LO 中取出送入通用寄存器中。更麻烦的是，由于流水线的问题，很多 MIPS 处理器对这些指令的使用还有些限制。这些新指令执行速度更快，同时也更容易使用。

● 多媒体指令的扩充：

参见龙芯多媒体指令手册。

● 定点操作使用浮点的数据通路：

在执行定点程序的过程中，浮点数据通路常常处于空闲状态，这些指令使得我们有机会利用它们，进一步增加指令并行的程度。

12.3 指令流

龙芯 2F 是一个多发射高度并行的处理器，对本质上是串行的指令流的处理可能会对程序性能产生明显的影响，本节讨论关于指令对齐、转移指令、指令调度等问题。

12.3.1 指令对齐

在一个周期内，龙芯 2F 可以从一个 Cache 行中取出四条指令，但这四条指令不能跨越 Cache 行的边界。我们应该对那些经常性被执行的基本块进行合适的对齐，以避免跨越 Cache 行的边界。此外，如果在一次所取的四条指令中存在转移指令，也会影响取指的效率。如果第一条是转移分支指令，而且转移预测是成功，那么最后两条指令将被抛弃。如果最后一条是转移指令，即使转移成功，处理器也不得不再取下一个 Cache 行以得到它的延迟槽中的指令。龙芯 2F 一个周期只能给一条转移指令译码，如果在一束指令中存在两条转移指令的话，它将需要两个周期来完成译码，也就是说取指引擎将被阻塞一个周期。

12.3.2 转移指令的处理

在龙芯 2F 的处理器中，指令流地址的一个意想不到的变化会浪费大约 10 条指令的时间。“意想不到” 可以是由转移成功的指令导致，也可能会由转移预测错误导致。对于目前的龙芯 2F 而言，即使是一个正确预测且预测转移成功的转移指令也比顺序代码慢，它会浪费一个周期，因为对于普通条件转移指令，转移目标缓存器（BTB）不能给出下一个正确的程序计数器 PC 值。

编译器可以通过以下的方法来减少转移指令引起的开销：

- 龙芯 2 号的转移指令预测方法和其它的高性能处理器都不同，且不同的版本都有一些细微的差别。基于执行剖析（Profile)，编译器可以根据实际的转移频率对代码位置进行重新安排，从而得到较好的预测结果。
- 尽可能使基本块变大。一种比较好的优化结果就是使得在两条转移成功的转移指令之间平均有 20 条指令。为了使它们之间至少含有 20 条指令，这就需要循环展开，还要把不到 20 条指令的子程序直接内联展开。龙芯 2F 实现了条件性移动指令，它可以用来减少分支指令数量。通过执行剖析来重新组织代码也有助于这个优化。

1.3 节给出了取指译码单元的一个概要描述。正如我们所见，不同的转移指令使用不同的方式进行预测：

- 静态预测。

  针对 likely 类转移指令和直接跳转指令。
- G-Share 预测器。

  一个 9 位的全局历史寄存器 GHR，和一个有 2K 项的模式历史表 PHT。用于条件转移指令。
- BTB（转移目标缓存）。

  有 16 项的全相联的缓存。被用于预测寄存器跳转指令的目标地址。
- RAS（返回地址栈）。

  4 项，被用于预测函数返回的目标地址。

以下有几点关于软件需要注意的地方：

在龙芯 2F 处理器上需要特别小心地使用 Likely 类转移指令。尽管 Likely 类转移指令也许对顺序标量处理器的简单的静态调度很有效，但是它对现代高性能处理器并不是同样有效。因为现代高性能处理器的转移预测硬件是比较复杂，它们通常有 90%以上的正确预测率。（比如说，龙芯 2F 能够正确预测 85%-100%，平均 95%的条件转移的转移方向）在这种情况下，编译器不应该使用预测率不太高的 Likely 类转移指令。事实上，我们发现带有-mno-branch-likely 选项的 GCC（3.3 版）通常会工作得更好。

取指译码单元被划分成 3 个流水段，其中转移的目标地址在第三阶段被计算。转移成功的转移指令将会导致有两个周期的停顿，也就是说，如果在周期 0 取出一条转移指令，在周期 1 取出地址为 PC+16 的指令，周期 2 取出地址为 PC+32 的指令，在周期 3 时，才会取到转移指令的目标地址。所以减少转移成功的转移指令数将会比较有一些帮助。

龙芯 2F 中的 BTB 仅被用于寄存器跳转指令(包括 JALR 及除 JR31 外的 JR 指令)。

通过一个 4 项的 RAS 来预测 JR31 指令的目标地址。函数返回的预测有效性取决于那些使用 JR31 指令作为函数返回指令的软件。

12.3.3 指令流密度的提高

编译器应该尽量利用执行剖析，以确保调入指令 Cache 的那些字节均被执行。这就要求跳转指令的目标地址需对齐，并且把那些很少被执行的代码移出 Cache 行。

12.3.4 指令调度

龙芯 2F 内部有比较大的指令窗口会进行动态的指令调度，但是由于处理器内部各种资源有限无法做到最优的调度，编译器可以在一定程度上协助处理器进行更好的调度。现代的编译器（如 GCC）有指令调度的支持，把龙芯 2F 内部的部件资源情况和指令的延迟情况告诉编译器，它就能够进行较好的调度。

12.4 存储器访问

Load/Store 指令的执行对整个系统性能有很大的影响。如果一级数据 Cache 中包括所需的内容，那么这些指令可以很快被执行。如果数据只在二级 Cache 则稍微慢些（需要增加额外的 11 拍），如果只在主存中则会有很大的延迟。不过，乱序执行和非阻塞 Cache 可以减少由这些延时带来的性能损失。

龙芯 2F 的二级 Cache 存放指令和数据，容量是 512KB，组织为四路组相联。它工作在处理器主频同样的频率，采用非阻塞结构，即每拍可以访问一次。龙芯 2F 内置 DDR2 内存控制器，最大限度地减少了内存访问的延迟。与龙芯 2E 不同，龙芯 2F 的内存工作频率是独立设置，与处理器主频没有关联，因此更有利于发挥内存的性能。有关内存控制器更多的信息可以参考第十章。

龙芯 2F 目前的这个版本没有直接提供预取指令，但是可以通过 Load-to-Zero-Register

来获得预取的功能，即向 0 号寄存器取一个数。0 号寄存器值永远为 0，因此这样的指令不会影响程序员可见的状态，但可以迫使一些数据进入 Cache。为了降低开销，这种指令执行的时候不会发生访问异常，即使地址非法也会被悄悄地忽略。

编译器应该尽量减少不必要的存储访问。目前的龙芯 2F 处理器的存储指令延迟较大（即使是 Cache 命中，也需要 5 个周期），同时指令窗口也没有大到可以容忍几十周期的访问延迟。

软件还要特别注意数据对齐的问题。集合体（数组，一些纪录，子程序堆栈帧）应该被分配在对齐的 Cache 行边界上，这样就可以利用 Cache 行对齐数据通路，还可以降低 Cache 行被填满的数目。在那些强迫不对齐（例如 GCC 的 Packed 属性）的集合体（纪录，普通块）中的项目中，应该产生一个编译时间的警告信息。在龙芯 2F 中正常的 Load/Store 指令有对齐要求，不满足要求的存储访问或者采用非对齐访存指令对来访问或者通过内核模拟来实现。例如，从非四字节对齐的地址取一个字（四字节）会触发例外，由操作系统来处理；通常操作系统需要几千个处理器周期才能完成这个任务。因此用户需要知道这些警告信息代表代码的性能可能会很低。编译参数的代码都默认这些参数是对齐的。那些经常被使用的标量应该驻留在寄存器中。

12.5 其他提示

- 使用所有的浮点寄存器。尽管 O32 ABI 只开放了 16 个给用户使用，但是龙芯 2F 提供了 32 个 64 位的浮点寄存器。使用 N32 或 N64 ABI 有助于发挥处理器的性能。
- 使用性能计数器。龙芯 2F 的性能计数器可以用来监控程序的实时性能参数。编译器和软件开发者可以通过分析这个结果来改进他们的代码。

13 龙芯 2F 与传统 MIPS 及 MIPS64 ISA 的差异

表 13-1 龙芯 2F 与传统 MIPS 及 MIPS64 ISA 的差异

关键字	Godson2F 特性	MIPS III	MIPS64
CP0 寄存器 $22	在进行 TLBWI 和 TLBWR 操作之后，龙芯 2F 的 ITLB 不会自动更新，而需要软件设置 CP0 22 号寄存器的 ITLB 位来 Flush ITLB。	MIPS III 无此特性，直接使用 TLBWI 和 TLBWR 即可。	MIPS 64 同 MIPS III
CP 0 寄存器 $24，$25	龙芯 2F 中 CP0 寄存器的 24 号和 25 号，被用做性能计数器，详情参见 5.19 章。	MIPS R4000 中，CP0 寄存器 24 号和 25 号保留。MIPS R10000 中，CP0 寄存器 24 号被保留。	在 MIPS64 中，CP0 寄存器 24 号被定义为 DEPC 寄存器，用于 EJTAG 。
CP 0 寄存器 $27	龙芯 2F 不支持 CacheErr，不支持 ERL 被置 1 时从 ErrEPC 中取返回地址，也不支持 ERL 置 1 时用户空间低 512M 为 Unmapped、Uncached。CP0 寄存器 27 号为保留寄存器。	MIPS R4000 与 R10000 都支持 Cache Error 例外。CP0 寄存器 27 号为 CacheErr 寄存器。ERL 位为 1 时从 ErrEPC 中取返回地址，且用户空间低 512M 为 Unmapped、Uncached。	MIPS 64 同 MIPS III
TLB 入口	龙芯 TLB 不支持不同大小的页共存。每次设置时，所有的 TLB 表项的页都只能被设置为相同大小。	MIPS R4000 与 R10000 每个 TLB 表项都能映射不同大小的页。	MIPS 64 同 MIPS III
地址错例外	龙芯 2F 屏蔽了向 0 号寄存器 Load 数据时的地址错例外，方便了编译时的指令预取。	MIPS III 中，向零号寄存器 Load 数据时，如果地址出错或者翻译失败，会立即引发一个地址错例外。	MIPS 64 同 MIPS III

浮点运算	龙芯 2F 浮点运算中，遇到负非数，负越界，负无穷时，如果 FCSR 寄存器的 Invalid Operation Enable 位没有被设置，将不会发出无效操作例外，返回值为负最大（0x80000000 或 0x80000000 00000000）。	MIPS III 的浮点运算中，遇到相同情况时，也不会发出无效操作例外，但返回值规定为正最大（0x7fffffff 或 0x7fffffff ffffffff）。	MIPS64 同 MIPS III
浮点控制状态寄存器	龙芯 2F 的 FCSR 寄存器无[CC2:CC7]这六位（[31:26]），只有 CC0、CC1 位可以用。这些位在龙芯中只读且始终为 0。	MIPS III 与龙芯 2F 相同	MIPS64 中这 7 位可读写。
TLB 例外	龙芯 2F 不支持 KX,UX,SX 位， XTLB 例外入口与 TLB 例外入口相同	MIPS III 中用 KX,UX,SX 位来区分 64 位地址和 32 位地址，且 TLB 例外和 XTLB 例外分别走两个例外入口	MIPS64 同 MIPS III

14 .参考代码

14.1 DDR2 SDRAM 参数配置举例

下面列举 PMON 中的 DDR2 配置程序

```
#define REG_ADDRESS 0x0
#define CONFIG_BASE 0xaffffe00

              .global ddr2_config
              .ent       ddr2_config
              .set       noreorder
              .set       mips3
ddr2_config:
       la           t0, ddr2_reg_data
       addu       t0, t0, s0
       li           t1, 0x1d                         //循环配置29个参数
       li           t2, CONFIG_BASE

reg_write:
       sd           a1, REG_ADDRESS(t2)
       subu        t1, t1, 0x1
       addiu        t0, t0, 0x8
       bne          t1, $0, reg_write
       addiu        t2, t2, 0x10

       //配置参数CTRL_03  中的start位
       li            t2, CONFIG_BASE
       la             t0,DDR2_CTL_start_DATA_LO
       addu         t0, t0, s0
       ld            a1, 0x0(t0)
       sd            a1, 0x30(t2)

       jr            ra
       nop
       .end        ddr2_config


       .rdata
       .align 5

ddr2_reg_data:
//0000000_0 arefresh 0000000_1 ap 0000000_1 addr_cmp_en 0000000_1 active_aging
DDR2_CTL_00_DATA_LO:   .word 0x00000101
// 0000000_1 ddrii_sdram_mode 0000000_1 concurrentap 0000000_1 bank_split_en 0000000_0
auto_refresh_mode
DDR2_CTL_00_DATA_HI:    .word 0x01000100 #no_concurrentap
//DDR2_CTL_00_DATA_HI:  .word 0x01010100
//0000000_0 ecc_disable_w_uc_err 0000000_1 dqs_n_en 0000000_0 dll_bypass_mode 0000000_0 dlllockreg
//DDR2_CTL_01_DATA_LO: .word 0x00010100 #dll_by_pass
DDR2_CTL_01_DATA_LO:   .word 0x00010000
//0000000_0 fwc 0000000_0 fast_write 0000000_0 enable_quick_srefresh 0000000_0 eight_bank_mode
DDR2_CTL_01_DATA_HI:    .word 0x00010000
//0000000_0 no_cmd_init 0000000_0 intrptwritea 0000000_0 intrptreada 0000000_0 intrptapburst
DDR2_CTL_02_DATA_LO:   .word 0x00000000
//0000000_1 priority_en 0000000_0 power_down 0000000_1 placement_en 0000000_1 odt_add_turn_clk_en
DDR2_CTL_02_DATA_HI:    .word 0x01000101
//0000000_1 rw_same_en 0000000_0 reg_dimm_enable 0000000_0 reduc 0000000_0 pwrup_srefresh_exit
```

```
DDR2_CTL_03_DATA_LO:   .word 0x01000000
//0000000_1 swap_port_rw_same_en 0000000_1 swap_en 0000000_0 start 0000000_0 srefresh
DDR2_CTL_03_DATA_HI:   .word 0x01010000
//0000000_0 write_modereg 0000000_1 writeinterp 0000000_1 tref_enable 0000000_1 tras_lockout
DDR2_CTL_04_DATA_LO:   .word 0x00010101
//000000_01 rtt_0 000000_00 ctrl_raw 000000_10 axi0_w_priority 000000_10 axi0_r_priority
DDR2_CTL_04_DATA_HI:   .word 0x01000202
//00000_100 column_size 00000_101 caslat 00000_010 addr_pins 000000_10 rtt_pad_termination
DDR2_CTL_05_DATA_LO:   .word 0x04050102 #CL =5
//DDR2_CTL_05_DATA_LO: .word 0x04040102 #CL =4
//00000_000 q_fullness 00000_000 port_data_error_type 00000_000 out_of_range_type 00000_000 max_cs_reg
DDR2_CTL_05_DATA_HI:   .word 0x00000000
//00000_010 trtp 00000_010 trrd 00000_010 temrs 00000_011 tcke
//DDR2_CTL_06_DATA_LO: .word 0x01020203 #125 M
//DDR2_CTL_06_DATA_LO: .word 0x02020203 #400
DDR2_CTL_06_DATA_LO:   .word 0x03050203 #800
//0000_1010 aprebit 00000_100 wrlat 00000_010 twtr 00000_100 twr_int
//DDR2_CTL_06_DATA_HI: .word 0x0a040203 #125 M
//DDR2_CTL_06_DATA_HI: .word 0x0a040204 #400
DDR2_CTL_06_DATA_HI:   .word 0x0a040306 #800
//0000_0000 ecc_c_id 0000_1111 cs_map 0000_0111 caslat_lin_gate 0000_1010 caslat_lin
//DDR2_CTL_07_DATA_LO: .word 0x000f0808 #CL=4//cs_map to cs0-cs3
DDR2_CTL_07_DATA_LO:   .word 0x00030a0b #CL=5//cs_map to cs0-cs3
//DDR2_CTL_07_DATA_LO: .word 0x000f0a0a #CL=5//cs_map to cs0-cs3
//DDR2_CTL_07_DATA_LO: .word 0x000c0708 #CL=4//cs_map to cs2-cs3
//0000_0000 max_row_reg 0000_0000 max_col_reg 0000_0010 initaref 0000_0000 ecc_u_id
//DDR2_CTL_07_DATA_HI: .word 0x00000200
DDR2_CTL_07_DATA_HI:   .word 0x00000400 #800
//0000_0001 odt_rd_map_cs3 0000_0010 odt_rd_map_cs2 0000_0100 odt_rd_map_cs1 0000_1000
odt_rd_map_cs0
DDR2_CTL_08_DATA_LO:   .word 0x01020408
//0000_0001 odt_wr_map_cs3 0000_0010 odt_wr_map_cs2 0000_0100 odt_wr_map_cs1 0000_1000
odt_wr_map_cs0
DDR2_CTL_08_DATA_HI:   .word 0x01020408
//0000_0000 port_data_error_id 0000_0000 port_cmd_error_type 0000_0000 port_cmd_error_id 0000_0000
out_of_range_source_id
DDR2_CTL_09_DATA_LO:   .word 0x00000000
//000_00000 ocd_adjust_pup_cs_0 000_00000 ocd_adjust_pdn_cs_0 0000_0100 trp 0000_1000 tdal
//DDR2_CTL_09_DATA_HI: .word 0x00000204 #125 M
//DDR2_CTL_09_DATA_HI: .word 0x00000408 #400
DDR2_CTL_09_DATA_HI:   .word 0x0000060c #800
//00_111111 age_count 000_01111 trc 000_00010 tmrd 000_00000 tfaw
//DDR2_CTL_10_DATA_LO: .word 0x3f070200 #125 M
//DDR2_CTL_10_DATA_LO: .word 0x3f0f0200 #400
DDR2_CTL_10_DATA_LO:   .word 0x3f1a021b #800
//0_0011101 dll_dqs_delay_2 0_0011101 dll_dqs_delay_1 0_0011101 dll_dqs_delay_0 00_111111
command_age_count
DDR2_CTL_10_DATA_HI:   .word 0x1717173f
//0_0011101 dll_dqs_delay_6 0_0011101 dll_dqs_delay_5 0_0011101 dll_dqs_delay_4 0_0011101
dll_dqs_delay_3
DDR2_CTL_11_DATA_LO:   .word 0x17171717
//0_1011111 wr_dqs_shift 0_1111111 dqs_out_shift 0_0011101 dll_dqs_delay_8 0_0011101 dll_dqs_delay_7
DDR2_CTL_11_DATA_HI:   .word 0x5f7f1517
//00001011 tras_min 00000000 out_of_range_length 00000000 ecc_u_synd 00000000 ecc_c_synd
//DDR2_CTL_12_DATA_LO: .word 0x05000000 #125 M
//DDR2_CTL_12_DATA_LO: .word 0x0b000000 #400
DDR2_CTL_12_DATA_LO:   .word 0x15000000 #800
//0000000_000101010 dll_dqs_delay_bypass_0 00011100 trfc 00000100 trcd_int
//DDR2_CTL_12_DATA_HI: .word 0x01ff0302 #125/2 M,read_dqs_delay_max
//DDR2_CTL_12_DATA_HI: .word 0x002a0302 #125/2 M
//DDR2_CTL_12_DATA_HI: .word 0x002a0602 #125 M
```

```
//DDR2_CTL_12_DATA_HI:  .word 0x002a3c04 #400
DDR2_CTL_12_DATA_HI:    .word 0x002a3c05 #800
//0000000_000101010 dll_dqs_delay_bypass_2 0000000_000101010 dll_dqs_delay_bypass_1
//DDR2_CTL_13_DATA_LO: .word 0x01ff01ff #read_dqs_delay_max
DDR2_CTL_13_DATA_LO:   .word 0x002a002a
//0000000_000101010 dll_dqs_delay_bypass_4 0000000_000101010 dll_dqs_delay_bypass_3
//DDR2_CTL_13_DATA_HI:  .word 0x01ff01ff #read_dqs_delay_max
DDR2_CTL_13_DATA_HI:    .word 0x002a002a
//0000000_000101010 dll_dqs_delay_bypass_6 0000000_000101010 dll_dqs_delay_bypass_5
//DDR2_CTL_14_DATA_LO: .word 0x01ff01ff #read_dqs_delay_max
DDR2_CTL_14_DATA_LO:   .word 0x002a002a
//0000000_000101010 dll_dqs_delay_bypass_8 0000000_000101010 dll_dqs_delay_bypass_7
//DDR2_CTL_14_DATA_HI:  .word 0x01ff01ff #read_dqs_delay_max
DDR2_CTL_14_DATA_HI:    .word 0x002a002a
//0000000_000000000 dll_lock 0000000_000100100 dll_increment
DDR2_CTL_15_DATA_LO:   .word 0x00000004
//0000000_010110100 dqs_out_shift_bypass 0000000_010000111 dll_start_point
DDR2_CTL_15_DATA_HI:    .word 0x00b40020
//000000_0000000000 int_ack 0000000_010000111 wr_dqs_shift_bypass
DDR2_CTL_16_DATA_LO:   .word 0x00000087
//00000_00000000000 int_status 00000_00000000000 int_mask
DDR2_CTL_16_DATA_HI:    .word 0x000007ff #no_interrupt
//DDR2_CTL_16_DATA_HI:  .word 0x00000000 #no_masked
//0_000000000000000 emrs1_data 00_00100000011011 tref
//DDR2_CTL_17_DATA_LO: .word 0x0000004b #125/16 M
//DDR2_CTL_17_DATA_LO: .word 0x0000009c #20 M
//DDR2_CTL_17_DATA_LO: .word 0x000004b0 #125 M
//DDR2_CTL_17_DATA_LO: .word 0x0000081b #400
DDR2_CTL_17_DATA_LO:   .word 0x0000101b #800
//0_000000000000000 emrs2_data_1 0_000000000000000 emrs2_data_0
DDR2_CTL_17_DATA_HI:    .word 0x00000000
//0_000000000000000 emrs2_data_3 0_000000000000000 emrs2_data_2
DDR2_CTL_18_DATA_LO:   .word 0x00000000
//0000000000011100 axi0_en_size_lt_width_instr 0_000000000000000 emrs3_data
DDR2_CTL_18_DATA_HI:    .word 0x001c0000
//0000000011001000 tdll 0000000001101011 tcpd
DDR2_CTL_19_DATA_LO:   .word 0x00c8006b
//0100100011100001 tras_max 0000000000000010 tpdex
//DDR2_CTL_19_DATA_HI:  .word 0x04b00002 #125 M
//DDR2_CTL_19_DATA_HI:  .word 0x48e10002 #400
DDR2_CTL_19_DATA_HI:    .word 0x68e10002 #800
//0000000011001000 txsr 0000000000011111 txsnr
//DDR2_CTL_20_DATA_LO: .word 0x00c8000f #125 M
//DDR2_CTL_20_DATA_LO: .word 0x00c8001f #400
DDR2_CTL_20_DATA_LO:   .word 0x00c8002f #800
//0000000000000000 xor_check_bits 0000000000000000 version
DDR2_CTL_20_DATA_HI:    .word 0x00000000
  //000000000000000000110110 tinit
DDR2_CTL_21_DATA_LO:   .word 0x00030d40 #real
//DDR2_CTL_21_DATA_LO: .word 0x00000036 #simulation
  //000_0000000000000000000000000000000000000 ecc_c_addr
DDR2_CTL_21_DATA_HI:    .word 0x00000000
//000000000000000000000000000_0000000000000000000000000000000000000 ecc_u_addr
DDR2_CTL_22_DATA_LO:   .word 0x00000000
DDR2_CTL_22_DATA_HI:    .word 0x00000000
//000000000000000000000000000_0000000000000000000000000000000000000 out_of_range_addr
DDR2_CTL_23_DATA_LO:   .word 0x00000000
DDR2_CTL_23_DATA_HI:    .word 0x00000000
//000000000000000000000000000_0000000000000000000000000000000000000 port_cmd_error_addr
DDR2_CTL_24_DATA_LO:   .word 0x00000000
DDR2_CTL_24_DATA_HI:    .word 0x00000000
```

```
//0000000000000000000000000000000000000000000000000000000000000000 ecc_c_data
DDR2_CTL_25_DATA_LO:  .word 0x00000000
DDR2_CTL_25_DATA_HI:   .word 0x00000000
//0000000000000000000000000000000000000000000000000000000000000000 ecc_u_data
DDR2_CTL_26_DATA_LO:  .word 0x00000000
DDR2_CTL_26_DATA_HI:   .word 0x00000000
//0000000000000000000000000000000000000000000000000000000000000000
DDR2_CTL_27_DATA_LO:  .word 0x00000000
DDR2_CTL_27_DATA_HI:   .word 0x00000000
//0000000000000000000000000000000000000000000000000000000000000000
DDR2_CTL_28_DATA_LO:  .word 0x00000001
DDR2_CTL_28_DATA_HI:   .word 0x00000000
//0000000_1 rw_same_en 0000000_0 reg_dimm_enable 0000000_0 reduc 0000000_0 pwrup_srefresh_exit
DDR2_CTL_start_DATA_LO: .word 0x01000000
//0000000_1 swap_port_rw_same_en 0000000_1 swap_en 0000000_0 start 0000000_0 srefresh
DDR2_CTL_start_DATA_HI: .word 0x01010100
```

14.2 CPU，PCI 窗口映射配置示例

14.2.1.cpu 复位后的默认配置:

```
3ff00000: 0000000000000000 0000000010000000
3ff00010: ffffffffff00000 ffffffffff00000
3ff00020: ffffffff0000000 ffffffff0000000
3ff00030: ffffffffff00000 ffffffffff00000
3ff00040: 0000000000000000 0000000010000001
3ff00050: 0000000000000000 0000000000000000
3ff00060: 0000000080000000 ffffffffff00000
3ff00070: ffffffffff00000 ffffffffff00000
3ff00080: fffffff80000000 ffffffffff00000
3ff00090: ffffffffff00000 ffffffffff00000
3ff000a0: 0000000000000000 0000000000000000
3ff000b0: 0000000000000000 0000000000000000
```

即:
Master0 窗口 0 的大小为 256M,映射到 DDR,CPU 的物理地址 0-256M,映射到 DDR 的 0-256M.
Master0 窗口 1 的大小为 256M,映射到 PCI,CPU 物理地址的 256M-512M 映射到 PCI，具体映射的方法由 PCIMAP 来决定。
Master0 窗口 3,4 禁掉了（因为 CPU 物理地址 40 位不在窗口 2,3 的范围内)。
Master1 窗口大小为 2G,映射到 DDR,PCI 2G-4G 映射到 DDR 的 0-2G。

14.2.1 master0(CPU 到 PCI 和 CPU 到 DDR)建议配置:

窗口 1 不建议更改，cpu 的启动地址也落在这个区域。256M 开始的 3 个 64M pci 窗口方便 32 位程序访问 pci 空间。
窗口 0 不能超过 256M,因为超过的话就和窗口 1 覆盖了。
将窗口 3 配置为 DDR，大小为实际的 DDR 大小,开始地址为 2G，最大 4G(这种配置方法最多支持 2G 的 DDR 空间)。

将窗口 4 配置为另外一个 1G PCI 窗口，CPU 的 1G-2G 到 PCI 的 1G-2G。

```
*(unsigned volatile long long *) 0x900000003ff00010 = 0x0000000080000000; //base 为 2G
*(unsigned volatile long long *) 0x900000003ff00030 =
0xffffffff00000000|mask;//0xffffffff80000000; //mask 为实际的 DDR 大小掩码,mask 一定为
~(2^n-1).
*(unsigned volatile long long *) 0x900000003ff00050 = 0; //map 到 DDR 的 0 地址
```

256M 开始的 3 个 64M PCI 窗口的分配:
第一个窗口建议固定映射到 0-0x03ffffff,因为 VGA 显存(0xb8000 开始)落在这个范围。
第二个窗口和第三个窗口建议配置成连续的 128M PCI 空间。为了防止 BIOS 和内核中 PCI 空间分配冲突，BIOS 中窗口 2 在 0x04000000-0x07ffffff,窗口 3 在 0x08000000-0x0bffffff。

```
/* 0,00 0010 ,0000 01,00 0000 */
*(volatile int *)0xffffffffbfe00110 = 0x02040; //PCIMAP
```

内核中的窗口 2 在 0x14000000-0x17ffffff,窗口 3 在 0x18000000-0x1bffffff。

```
/* 0,00 0110 ,0001 01,00 0000 */
*(volatile int *)0xffffffffbfe00110    = 0x06140; //PCIMAP
```

注意这样 BIOS 中访问 PCI 设备的 CPU 物理地址和 PCI 地址是不同的,PCI 地址或 0x10000000 才是 CPU 访问这段 PCI 区域的物理地址。
内核中 CPU 访问 PCI 地址和 PCI 设备被分配的地址相同(这也是照顾目前内核中驱动的写法和驱动程序移植的方便)。
下面是内核中将窗口 4 配置成 CPU 1G-2G 到 PCI 1G-2G 的映射。

```
        loongson2e_pci_mem_resource.start    = 0x40000000UL;
        loongson2e_pci_mem_resource.end      = 0x7fffffffUL;
        /*set cpu window3 to map CPU 1G-> PCI 1G */
      asm(".set mips3;dli $2,0x900000003ff00000;li $3,0x40000000;sd $3,0x18($2);or
$3,1;sd $3,0x58($2);dli $3,0xffffffffc0000000;sd $3,0x38($2);.set mips0" :::"$2","$3");
```

14.2.2 **Master1(pci 到 ddr 映射)建议配置：**

PCI 到 DDR 映射牵涉到 PCI 配置头部分寄存器和地址窗口寄存器两部分。
PCI 配置头部分寄存器设置什么地址范围内的 PCI 访问能进入 CPU。Master1 地址窗口寄存器决定进来的地址映射到 DDR 的什么地址上。配置头部分寄存器包括基地址和 Mask 寄存器。
建议 Master1 窗口 0 为 2G，PCI 2G－4G 直接对应于 2G DDR 空间。
建议 Master1 窗口 1 为 8M,8-16M 主要用于软驱等设备 DMA 使用 (注意不能位于 0M，因为和 VGA 显存地址冲突，冲突的时候 CPU 访问 VGA 显存地址的时候会同时写到 DDR 和 VGA 显存里面)。
Master1 窗口 3，4 目前没有用到。

```
/*
 * PCI to local mapping: [2G,2G+256M] -> [0,256M]
 */
*(volatile int *)0xffffffffbfe00010 = 0x80000000; //PCI_BASE0 基址为 2G
*(volatile int *)0xffffffffbfe00014 = 0x0;
*(volatile int *)0xffffffffbfe00150= 0x8000000c; //大小为 2G,PCI64,预取使能,允许突发传输
*(volatile int *)0xffffffffbfe00154 = 0xffffffff;

/*set pci 2G -> DDR 0 ,window size 2G*/
   asm(".set mips3;dli $2,0x900000003ff00000;li $3,0x80000000;sd $3,0x60($2);sd $0,0xa0($2);dli $3,0xffffffff80000000;sd $3,0x80($2);.set mips0" :::"$2","$3");

/*
 * PCI to local mapping: [8M,16M] -> [8M,16M]
 */
*(volatile int *)0xffffffffbfe00018= 0x00800000;//PCI_BASE1 基址为 8M
*(volatile int *)0xffffffffbfe0001c = 0x0;
*(volatile int *)0xffffffffbfe00058 = 0xff80000c;//大小 8M
*(volatile int *)0xffffffffbfe0005c = 0xffffffff;

/*set pci 8-16M -> DDR 8-16M ,window size 8M*/
   asm(".set mips3;dli $2,0x900000003ff00000;li $3,0x800000;sd $3,0x68($2);sd $3,0xa8($2);dli $3,0xffffffffff800000;sd $3,0x88($2);.set mips0" :::"$2","$3");
```

14.3 PCI 配置空间访问方法

```
//pci_conf_read(int bus,int addr,int access_type)
#define PCI_ACCESS_READ    0
if (bus == 0) {
    /* Type 0 configuration on onboard PCI bus */
    if (device > 20 || function > 7)
        return ~0;          /* device out of range */
    addr = (1 << (device+11)) | (function << 8) | reg;
    type = 0x00000;
    }
    else {
    /* Type 1 configuration on offboard PCI bus */
    if (bus > 255 || device > 31 || function > 7)
```

```
        return ~0;    /* device out of range */
    addr = (bus << 16) | (device << 11) | (function << 8) | reg;
    type = 0x10000;
    }
*(volatile int *)0xffffffffbfe00004 |= 0x28000000;//清 PCICMD 的 Master Abort 和 Target Abort
*(volatile int *)0xffffffffbfe00118 = (addr >> 16) | type; //PCIMAP_CFG
if(access_type==PCI_ACCESS_READ)
data = *(volatile pcireg_t *) 0xffffffffbfe80000| (addr & 0xfffc));
else
*(volatile pcireg_t *) 0xffffffffbfe80000| (addr & 0xfffc));=data;
```

14.4 PCI IO 访问方法

```
//outb(data,port);
*(volatile char *)(0xffffffffbfd00000+port)=data;
//data=inb(port);
data=*(volatile char *)(0xffffffffbfd00000+port);
```

14.5 中断设置

CPU 引脚的 int[3:0]对应 cause 寄存器的 IP5~IP2,cpu 内部中断控制器对应 case 寄存器的 IP6,cp0_counter 比较寄存器中断(时钟中断)对应 cause 寄存器的 IP7。
当中断发生时触发通用异常，异常处理函数中读 cause 寄存器中的 EXCODE 为 0，判断异常原因是中断，然后读 cause 寄存器中的 IP2-IP7 判断中断是来自什么那个引脚。
IP2-IP7 的中断屏蔽参考 CP0_STATUS 寄存器。IP6 内部中断源屏蔽参考中断控制器一章。所有中断源电平的触发方式都可以设置，参考中断控制器一章。
因为中断源通过软件寄存器查询得到，因此中断号也完全由软件决定。

14.6 PCI 仲裁设置

某些 PCI 设备占用总线的时候，如果其他设备优先级别高于当前设备而打断其操作的时候会造成该设备工作异常。必须等到这些设备主动撤掉请求，别的设备才能占用总线。

```
//can not change gnt to break pci transfer when device's gnt not deassert for some broken device
    *(volatile int *)0xffffffffbfe00168=0x00fe0105;
```

当 CPU 猜测执行指令预取的时候，可能产生对 PCI 空间的读操作，有的 PCI 设备某一段空间不允许访问，如果访问就直接返回 retry 命令。这个时候为了防止 CPU 无限期 retry,可以设置最大 retry 次数寄存器。

```
//make pci retry max 32.
*(volatile int *)0xffffffffbfe00058 |=0x2000;
```

14.7 视频加速设置

下面是一段 mplayer 中通过 VIDIX 显示驱动方式利用视频加速功能进行视频播放的例子（主要函数）。更多的关于 VIDIX 请参见 VIDeo interface for *niX 项目（http://vidix.sf.net）及其相关文档：

```
static int is_supported_fourcc(uint32_t fourcc) /* 视频加速模块支持的色彩空间格式 */
{
    switch(fourcc)
     {
     case IMGFMT_YV12:
         return supports_planar;
     default:
         return 0;
     }
}


/* 显示初始化代码，主要进行相关寄存器值的计算等，寄存器含义及其使用方法请参见视频加速寄存器定义 */
static int godson_vid_init_video( vidix_playback_t *config )
{
    godson_vid_stop_video();
/* warning, if left or top are != 0 this will fail, as the framesize is too small then */
     left = config->src.x;
     top =   config->src.y;
     src_h = config->src.h;
     src_w = config->src.w;
     is_420 = 0;
     if(config->fourcc == IMGFMT_YV12 ||
         config->fourcc == IMGFMT_I420 ||
```

```
       config->fourcc == IMGFMT_IYUV) is_420 = 1;

 dest_w = config->dest.w;
 dest_h = config->dest.h;
 besr.fourcc = config->fourcc;

config->offset.y = 0;
config->offset.u = 0x20;
config->offset.v = 0x40;

src_offset_y = 0;
src_offset_u = 0x20;
src_offset_v = 0x40;

 num_godson_buffers= config->num_frames;

godson_buffer_base[0][0]= (godson_overlay_offset + src_offset_y);
godson_buffer_base[0][1]= (godson_overlay_offset + src_offset_u);
godson_buffer_base[0][2]= (godson_overlay_offset + src_offset_v);

/* godson paramter cal begin */

outFMT = 0x0;
inFMT = 0x1;
ZoomEn = 0x1;
Y2R_EN = 0x1;
res = 0xa;

ori_x = src_w;
ori_y = src_h;
out_x = dest_w;
out_y = dest_h;
```

```
framebuffer = 0x14000000 + (config->dest.y) * 1280*2 + (config->dest.x) * 2;
stride = 1280*2;

//end setting

reg0 = 0;
reg0 = reg0|(((unsigned int)Y2R_EN) << 1);
reg0 = reg0|(((unsigned int)ZoomEn) << 2);
reg0 = reg0|(((unsigned int)inFMT) << 3);
reg0 = reg0|(((unsigned int)outFMT) << 5);
reg0 = reg0|(((unsigned int)res) << 7);

reg1 = 0;
reg1 = reg1|(((unsigned int)ori_y));
reg1 = reg1|(((unsigned int)ori_x)<< 11);

reg2 = 0;
reg2 = reg2|(((unsigned int)out_y));
reg2 = reg2|(((unsigned int)out_x)<< 11);

reg3 = framebuffer;

reg4 = stride;

ov_stepx = FixFloat(ori_x)/ out_x;
ov_stepy = FixFloat(ori_y)/ out_y;

reg5 = 0;
reg5 = reg5|(((unsigned int)ov_segment_size));
reg5 = reg5|(((unsigned int)ov_stepx)<< 11/*23*/);

ov_size_mul_step = ov_stepx * (ov_segment_size-1);
```

```
        if(ori_x%64 == 0)
                ov_last_segment_size = cal_actual_point(FixFloat(ori_x), ov_stepx) - cal_actual_point(FixFloat(ori_x - 64), ov_stepx);
        else
                ov_last_segment_size = cal_actual_point(FixFloat(ori_x), ov_stepx) - cal_actual_point(FixFloat((ori_x/64)*64), ov_stepx);
    /*  if(ori_x%32 == 0){
            ov_last_segment_size = (FixFloat(ori_x) / ov_stepx) - (FixFloat(ori_x - 32)/ov_stepx);
        }else{
            ov_last_segment_size = (FixFloat(ori_x) / ov_stepx) - (FixFloat((ori_x / 32)*32)/ov_stepx);
        }
    */

        reg6 = 0;
        reg6 = reg6|(((unsigned int)ov_last_segment_size));
        reg6 = reg6|(((unsigned int)ov_size_mul_step)<< (11/*+12*/));

        reg7 = 0;
        reg7 = reg7|(((unsigned int)(ov_stepy))/*<< 12*/);

        reg8 = config->dest.x;
        reg9 = 1280;

         return 0;
    }

    /* 将 godson_vid_init_video 函数计算得到的寄存器值写入相应寄存器，开始视频回放 */
    static void godson_vid_display_video( void )
    {
        *((unsigned int *)godson_mmio_base + 0) = reg0 | 0x1;          //control
        *((unsigned int *)godson_mmio_base + 1) = reg1;        //ori pic size
```

```
        *((unsigned int *)godson_mmio_base + 2) = reg2;         //out pic size
        *((unsigned int *)godson_mmio_base + 3) = reg3;         //framebuffer start addr
        *((unsigned int *)godson_mmio_base + 4) = reg4;         //stride
        *((unsigned int *)godson_mmio_base + 5) = reg5;         //zoom control 1
        *((unsigned int *)godson_mmio_base + 6) = reg6;         //zoom control 2
        *((unsigned int *)godson_mmio_base + 7) = reg7;         //zoom control 3
        *((unsigned int *)godson_mmio_base + 8) = reg8;         //zoom control 3
        *((unsigned int *)godson_mmio_base + 9) = reg9;         //zoom control 3

        *((unsigned int *)godson_mmio_base + 0) = reg0 & 0xfffffffe;          //control
    }
```

为得到更加明显的加速效果，播放程序需要使用 Uncached Accelerated 的 TLB 映射。设置方法可以在内核代码 drivers/char/mem.c 文件中修改 phys_mem_access_prot()函数，检测是否是 Framebuffer 地址空间，如果是就附加属性 _CACHE_UNCACHED_ACCELERATED。通过这样的方法可以打开龙芯 2F 的 Uncache 加速功能。

打开 uncache 加速的例子：

linux 内核 driver/char/mem.c

```
#ifdef CONFIG_LOONGSON_VIDEO_ACCELERATED
static unsigned long vgamem_start=0,vgamem_end=0;
static int videoacc=1;
#endif
static int mmap_mem(struct file * file, struct vm_area_struct * vma)
{
        size_t size = vma->vm_end - vma->vm_start;

        if (!valid_mmap_phys_addr_range(vma->vm_pgoff, size))
                return -EINVAL;

        vma->vm_page_prot = phys_mem_access_prot(file, vma->vm_pgoff,
                                                 size,
                                                 vma->vm_page_prot);
#ifdef CONFIG_LOONGSON_VIDEO_ACCELERATED
        if(videoacc)
        {
        unsigned long offset = vma->vm_pgoff << PAGE_SHIFT;
                if((offset >= vgamem_start) && (offset <vgamem_end))
                vma->vm_page_prot =
__pgprot((pgprot_val(vma->vm_page_prot)&~_CACHE_MASK)|_CACHE_UNCACHED_ACC
```

```
ELERATED);
        }
#endif

        /* Remap-pfn-range will mark the range VM_IO and VM_RESERVED */
        if (remap_pfn_range(vma,
                        vma->vm_start,
                        vma->vm_pgoff,
                        size,
                        vma->vm_page_prot))
                return -EAGAIN;
        return 0;
}

#ifdef CONFIG_LOONGSON_VIDEO_ACCELERATED
#include <linux/pci.h>
#ifndef pci_for_each_dev
#define pci_for_each_dev for_each_pci_dev
#endif

static int __init myvgamem_init(void)
{
struct pci_dev *dev=0;
struct resource *r;
int idx;
if(!videoacc)return 0;
pci_for_each_dev(dev) {
if ((dev->class >> 16) == PCI_BASE_CLASS_DISPLAY)
{
       for (idx=0; idx < PCI_NUM_RESOURCES; idx++) {

              r = &dev->resource[idx];
              if (!r->start && r->end) {
                     continue;
              }
              if (r->flags & IORESOURCE_IO)
              continue;
              if (r->flags & IORESOURCE_MEM)
              {
                vgamem_start=r->start;
                vgamem_end=r->end;
                printk("vga:start=%lx,end=%lx\n",vgamem_start,vgamem_end);
                return 0;
              }
       }

}
}
printk("<0>can not find vga device\n");
  return 0;
}
```

```
late_initcall(myvgamem_init);
static int __init videoacc_setup(char *options)
{
        if (!options || !*options)
                return 0;
        if(options[0]=='0')videoacc=0;
        else videoacc=1;
        return 1;
}

__setup("videoacc=", videoacc_setup);
#endif
```

15 Errata:乱序执行的问题

龙芯 2F 处理器核在猜测执行时有可能到一些意想不到的地方（特别是 IO 区域）取指执行，这通常是不允许的。处理器选择转移目标 PC 有两种途径：一是根据转移历史记录作猜测；二是由功能部件运算结果提供。龙芯 2 号处理器的指令窗口中最多可处理 8 条转移指令，因此一条间接转移指令的目标地址可能在前面还有其它转移的情况下算出结果并送往取指部件。这样就有可能在核心态下发出对 IO 区域的 4 字节读操作。

为了避免猜测读到危险的地址区域，可以在操作系统中下加入下列特殊处理：

1. 从用户态切换到核心态时刷 BTB 及 RAS，以去除用户态下残留的历史记录。
2. 合理分配核心态下代码与 IO 的地址，以便在核心态下每条间接转移指令前都加上对目标 PC 的冗余处理。保证即使在错误的执行路径上也不至于跳到 IO 的地址。

附录 A 龙芯新的整型指令

MULTG —乘(Godson2)

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
SPECIAL2 011100	**rs**	**rt**	**rd**	**0 00000**	**MULTG 010000**
6	5	5	5	5	6

指令格式: MULTG rd, rs, rt

指令功能: 32 位有符号整数乘。

指令描述: rd ← rs * rt

通用寄存器rs中32位值乘以通用寄存器rt中32位值，这两个操作数都是有符号数，产生一个64位结果。结果的低32位保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

prod ← GPR[rs]$_{31..0}$ * GPR[rt]$_{31..0}$

rd ← sign_extend(prod$_{31..0}$)

例外:

无



MULTUG —无符号乘(Godson2)

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
SPECIAL2 **011100**	**rs**	**rt**	**rd**	**0** **00000**	**MULTUG** **010010**
6	5	5	5	5	6

指令格式: MULTUG rd, rs, rt

指令功能: 32 位无符号整数乘。

指令描述: rd ← rs * rt

通用寄存器rs中32位值乘以通用寄存器rt中32位值，这两个操作数都是无符号数，产生一个64位结果。结果的低32位保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

prod ← (0 || GPR[rs]$_{31..0}$) * (0 || GPR[rt]$_{31..0}$)

rd ← sign_extend(prod$_{31..0}$)

例外:

无



DMULTG —双字乘(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 **011100**	**rs**	**rt**	**rd**	**0** **00000**	**DMULTG** **010001**
6	5	5	5	5	6

指令格式: DMULTG rd, rs, rt

指令功能: 64 位有符号整数乘。

指令描述: rd ← rs * rt

通用寄存器rs中64位值乘以通用寄存器rt中64位值，这两个操作数都是有符号数，产生一个128位结果。结果的低64位保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

```
prod ← GPR[rs] * GPR[rt]
rd ← prod63..0
```

例外:

无



DMULTUG —无符号双字乘(Godson2)

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
SPECIAL2 **011100**	**rs**	**rt**	**rd**	**0** **00000**	**DMULTUG** **010011**
6	5	5	5	5	6

指令格式: DMULTG rd, rs, rt

指令功能: 64 位无符号整数乘。

指令描述: rd ← rs * rt

通用寄存器rs中64位值乘以通用寄存器rt中64位值，这两个操作数都是无符号数，产生一个128位结果。结果的低64位保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

prod ← (0 || GPR[rs]) * (0 || GPR[rt])

rd ← $prod_{63..0}$

例外:

无



DIVG —除(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	**rs**	**rt**	**rd**	**0 00000**	**DIVG 010100**
6	5	5	5	5	6

指令格式: DIVG rd, rs, rt

指令功能: 32 位有符号整数除。

指令描述: rd ← rs / rt

通用寄存器rs中32位值除以通用寄存器rt中32位值，这两个操作数都是有符号数。32位商保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

q ← GPR[rs]$_{31..0}$ div GPR[rt]$_{31..0}$

LO ← sign_extend(q$_{31..0}$)

例外:

无



DIVUG —无符号除(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs	rt	rd	0 00000	DIVUG 010110
6	5	5	5	5	6

指令格式: DIVUG rd, rs, rt

指令功能: 32 位无符号整数除。

指令描述: rd ← rs / rt

通用寄存器rs中32位值除以通用寄存器rt中32位值，这两个操作数都是无符号数。32位商保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

q ← (0 || GPR[rs]$_{31..0}$) div (0 || GPR[rt]$_{31..0}$)

rd ← sign_extend(q$_{31..0}$)

例外:

保留指令



DDIVG —双字除(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	**rs**	**rt**	**rd**	**0 00000**	**DDIVG 010101**
6	5	5	5	5	6

指令格式: DDIVG rd,rs, rt

指令功能: 64 位有符号整数除。

指令描述: rd ← rs / rt

通用寄存器rs中64位值除以通用寄存器rt中64位值，这两个操作数都是有符号数。64位商保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

rd ← GPR[rs] div GPR[rt]

例外:

无



DDIVUG —无符号双字除(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	**rs**	**rt**	**rd**	**0 00000**	**DDIVUG 010111**
6	5	5	5	5	6

指令格式: DDIVUG rd, rs, rt

指令功能: 64 位无符号整数除。

指令描述: rd ← rs / rt

通用寄存器rs中64位值除以通用寄存器rt中64位值，这两个操作数都是无符号数。64位商保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

rd ← (0 || GPR[rs]) div (0 || GPR[rt])

例外:

无

MODG —求模(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs	rt	rd	0 00000	MODG 011100
6	5	5	5	5	6

指令格式: MODG rd, rs, rt

指令功能: 32 位有符号整数求模。

指令描述: rd ← rs % rt

通用寄存器rs中32位值除以通用寄存器rt中32位值，这两个操作数都是有符号数。32位剩余数保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

q ← GPR[rs]$_{31..0}$ mod GPR[rt]$_{31..0}$

HI ← sign_extend(q$_{31..0}$)

例外:

无

MODUG —无符号求模(Godson2)

31　　26	25　　21	20　　16	15　　11	10　　6	5　　0
SPECIAL2 **011100**	**rs**	**rt**	**rd**	**0** **00000**	**MODUG** **011110**
6	5	5	5	5	6

指令格式: MODUG rd, rs, rt

指令功能: 32 位无符号整数求模。

指令描述: rd ← rs % rt

通用寄存器rs中32位值除以通用寄存器rt中32位值，这两个操作数都是无符号数。32位剩余数保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

q ← (0 || GPR[rs]$_{31..0}$) mod (0 || GPR[rt]$_{31..0}$)

rd ← sign_extend(q$_{31..0}$)

例外:

保留指令

DMODG —双字求模(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	**rs**	**rt**	**rd**	**0 00000**	**DMODG 011101**
6	5	5	5	5	6

指令格式: DMODG rd, rs, rt

指令功能: 64 位有符号整数求模。

指令描述: rd ← rs % rt

通用寄存器rs中64位值除以通用寄存器rt中64位值，这两个操作数都是有符号数。64位剩余数保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

rd ← GPR[rs] mod GPR[rt]

例外:

无

DMODUG —无符号双字求模(Godson2)

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs	rt	rd	0 00000	DMODUG 011111
6	5	5	10		6

指令格式: DMODUG rd, rs, rt

指令功能: 64 位无符号整数求模。

指令描述: rd ← rs % rt

通用寄存器rs中64位值除以通用寄存器rt中64位值，这两个操作数都是无符号数。64位剩余数保存在特殊寄存器rd中。

任何情况下都不会产生算术异常。

操作:

rd ← (0 || GPR[rs]) mod (0 || GPR[rt])

例外:

无

附录 B 龙芯新的浮点指令

表 B-1 龙芯 2 号浮点单元的 Paired-single(PS) 指令

OP \ Fmt	Fmt=22
ADD	Add.ps
SUB	Sub.ps
NEG	Neg.ps
ABS	Abs.ps
C.F	C.F.ps
C.UN	C.UN.ps
C.EQ	C.EQ.ps
C.UEQ	C.UEQ.ps
C.OLT	C.OLT.ps
C.ULT	C.ULT.ps
C.OLE	C.OLE.ps
C.ULE	C.ULE.ps
C.SF	C.SF.ps
C.NGLE	C.NGLE.ps
C.SEQ	C.SEQ.ps
C.NGL	C.NGL.ps
C.LT	C.LT.ps
C.NGE	C.NGE.ps
C.LE	C.LE.ps
C.NGT	C.NGT.ps
MUL	MUL.ps
MOV	MOV.ps



MADD.fmt—浮点乘加

31　　26	25　　21	20　　16	15　　11	10　　6	5　　0
SPECIAL2 011100	**Fmt**	**ft**	**fs**	**fd**	**MADD 011000**
6	5	5	5	5	6

指令格式:

MADD.S fd, fs, ft

MADD.D fd, fs, ft

指令功能: 浮点值的先乘后加。

指令描述: fd ← ((fs * ft) + fd)

先将浮点寄存器ft中的值乘以浮点寄存器fs中的值，得到一个乘积。再把这个乘积加上浮点寄存器fd中的值，得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照FCSR的当前进位模式，结果保存进fd中。操作数和运算结果都是fmt格式。

操作:

```
vfd ← ValueFPR(fd, fmt)
vfs ← ValueFPR(fs, fmt)
vft ← ValueFPR(ft, fmt)
StoreFPR(fd, fmt, vfd + vfs * vft)
```

例外:

不可用协处理器例外

保留指令例外

浮点：

不精确例外	未实现操作例外
无效操作例外	上溢
下溢	

MSUB.fmt—浮点乘减

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	**Fmt**	**ft**	**fs**	**fd**	**MSUB 011001**
6	5	5	5	5	6

指令格式:

MSUB.S fd, fs, ft

MSUB.D fd, fs, ft

指令功能： 浮点值的先乘后减。

指令描述: fd ← (fs * ft) - fd

先将浮点寄存器ft中的值乘以浮点寄存器fs中的值，得到一个乘积。再把这个乘积减去浮点寄存器fd中的值，得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照FCSR的当前进位模式，结果保存进fd中。操作数和运算结果都是fmt格式。

操作:

```
vfd ← ValueFPR(fd, fmt)
vfs ← ValueFPR(fs, fmt)
vft ← ValueFPR(ft, fmt)
StoreFPR(fd, fmt, (vfs * vft)-vfd)
```

例外:

不可用协处理器例外

保留指令例外

浮点：

不精确例外	未实现操作例外
无效操作例外	上溢
下溢	

NMADD.fmt—浮点数乘加取负

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	**Fmt**	**ft**	**fs**	**fd**	**NMADD 011010**
6	5	5	5	5	6

指令格式:

NMADD.S fd, fs, ft

NMADD.D fd, fs, ft

指令功能: 对先乘后加的结果取负。

指令描述: fd ←- ((fs * ft) + fd)

先将浮点寄存器ft中的值乘以浮点寄存器fs中的值，得到一个乘积。再把这个乘积加上浮点寄存器fd中的值，再把和值取负得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照FCSR的当前进位模式，结果保存进fd中。操作数和运算结果都是fmt格式。

操作:

```
vfd ← ValueFPR(fd, fmt)
vfs ← ValueFPR(fs, fmt)
vft ← ValueFPR(ft, fmt)
StoreFPR(fd, fmt, -(vfd + vfs * vft))
```

例外:

不可用协处理器例外

保留指令例外

浮点：

不精确例外　　未实现操作例外

无效操作例外　　上溢

下溢

NMSUB.fmt—浮点数的乘减取负

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	Fmt	ft	fs	fd	NMSUB 011011
6	5	5	5	5	6

指令格式:

NMSUB.S fd, fs, ft

NMSUB.D fd, fs, ft

指令功能： 对先乘后减的结果取负。

指令描述: fd ← -((fs * ft) - fd)

先将浮点寄存器ft中的值乘以浮点寄存器fs中的值，得到一个乘积。再把这个乘积减去浮点寄存器fd中的值，再把差值取负得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照FCSR的当前进位模式，结果保存进fd中。操作数和运算结果都是fmt格式。

操作:

```
vfd ← ValueFPR(fd, fmt)
vfs ← ValueFPR(fs, fmt)
vft ← ValueFPR(ft, fmt)
StoreFPR(fd, fmt, -((vfs * vft)-vfd))
```

例外:

不可用协处理器例外

保留指令例外

浮点：

不精确例外　　未实现操作例外

无效操作例外　　上溢

下溢

修订历史

下表记录本文档的修订历史。

日期	版本	修订内容
2008 年 8 月	1.0	初始发行版本